SCHOOL OF LAW
CASE WESTERN RESERVE UNIVERSITY

Case Western Reserve Law Review

Volume 58 | Issue 2

2007

Manuscript: You Can't Patent Software: Patenting Software is Wrong

Peter D. Junger

MANUSCRIPT*

YOU CAN'T PATENT SOFTWARE: PATENTING SOFTWARE IS WRONG

Peter D. Junger†

INTRODUCTION

Until the invention of programmable[1] digital computers around the time of World War II, no one had imagined—and probably no one could have imagined—that methods of solving mathematical

---

* *Editor's Note: This article is the final known manuscript of Professor Peter Junger. We present this piece to you as a tribute to Professor Junger and for your own enjoyment. This piece was not, at the time of Professor Junger's passing, submitted to any Law Review or legal journal. Accordingly, Case Western Reserve University Law Review is publishing this piece as it was last edited by Professor Junger, with the following exceptions: we have formatted the document for printing, and corrected obvious typographical errors. Footnotes have been updated to the best of our ability, but without Professor Junger's input, you may find some errors. Internal references are likely to be inconsistent and could not be corrected without the author's input; internet sources may be out of date or unavailable. The article was originally published as a work-in-progress at Professor Junger's website, http://samsara.law.cwru.edu/patart/index.html (now unavailable).*

† I want to thank Judith Kaul for all the assistance that she has given me over the years in locating the materials that made writing this article possible. I also want to express my great debt of gratitude to Gino J. Scarselli for all the uncompensated work that he did in establishing that the First Amendment protects the publication of software in the case of *Junger v. Daley*. And finally I must apologize to Flanders and Swan for appropriating—more or less—a small portion of the lyrics of You Can't Eat People for the title of this article.

[1] Among the early programmable digital computers were Konrad Zuse's Z3 developed in Germany in 1941, the Colossus (which was not fully programmable, not being Turing complete) developed by Tommy Flowers in England in 1943, and Mauchly and Eckert's ENIAC I completed in 1946 in the United States. It is also worth noting that Charles Babbage designed, but was unable to actually construct, a programmable—using punch cards—mechanical computer, called the "analytical engine," back in the second-half of the nineteenth century. *See infra* Part VI.A.3(a)(i). Augusta Ada King, Countess of Lovelace wrote a program for Babagge's Analytical Engine and thus is usually credited with being the world's first computer programmer. *Id.* Both Babbage and Lady Lovelace were skilled mathematicians.

problems could be patented. In 1972, however, the Supreme Court of the United States[2] was confronted in *Gottschalk v. Benson*[3] with the question of whether "an application for [a patent on] an invention . . . related 'to the processing of data by program and more particularly to the programmed conversion of numerical information'[4] in general-purpose digital computers,"[5] covered patentable subject matter; the Court held that the claimed invention was not patentable,[6] in effect holding that most, and probably all, claims that relate to programs for the processing of data—that is, most claims that relate to what is commonly, but perhaps unfortunately,[7] called "software"—do not cover patentable subject matter.[8]

Since then the holding in *Benson* has not been overruled—or even questioned—by the Court, yet it appears today that most patent lawyers, and most legal academics who think about such matters, are now convinced that software is patentable and that *Benson* is no longer good law having, in effect. been overruled by later decisions of a lower court, the Federal Court of Appeals for the Federal Circuit.[9]

It is, however, my contention in this article that *Benson* is still good law—after all the Supreme Court is not likely to concede that its opinions can be overruled by any of the lower courts—and, more importantly from an academic point of view, that *Benson* was correctly decided, for most of the arguments for overturning the holding in *Benson* and permitting the granting of patents on software rest on a fundamental misunderstanding of the nature of the information that is processed by—and that is all that can be processed by—a computer.

Thus it is my contention here that inventions, no matter how novel or useful they may be, in the field of computer programming—that is,

---

[2] In this article I am only concerned with the law of the United States. If I know little about U.S. patent law, I know infinitely less about that of other jurisdictions.

[3] 409 U.S. 63 (1972).

[4] All that computer programs do can fairly be described as the conversion—*i.e.*, the processing—of information. *See infra* Part VI.C.

[5] 409 US at 64.

[6] In this article I argue at length—and, frankly, I think quite persuasively—that the holding in *Benson* that the application in that case did not cover a patentable invention applies to all applications for patents on inventions that consist of computer programs for the processing of data—and that all computer programs do nothing more, or less, than control the way in which a computer processes data, *i.e.* numerical information." For a discussion of *Benson* in some detail, *see infra* Part II.A.

[7] *See infra* note 13 and accompanying text.

[8] In *Benson* the Court was careful to say: "It is said that the decision precludes a patent for any program servicing a computer. We do not so hold", 409 U.S. at 71, but that cautious statement does not, of course, amount to a holding, or even a dictum, that there are programs that are patentable.

[9] *See infra* Part III.

in the field of processing data or information—are not and should not be patentable.

I. COMPUTERS AND SOFTWARE

We all know what computers are, or at least we think we do: they are machines[10] that process information.[11] Few of us, on the other hand, have any idea of what information is,[12] even though we are supposedly living in an "information age." Moreover, few of us have any clear idea about what software is other than the generic name for computer programs.[13] And most of us, I fear, do not really understand what computer programs are—or what, for that matter, they are not.[14]

A quick, but perhaps not too informative, definition of software is:

> Data that controls how a computer processes data.[15]

Computers, on the other hand, are tangible, kickable gadgets that will in a pinch serve as doorstops[16] and that are designed to be used in the processing of information.[17] If someone were to invent a new type of computer or a new way to manufacture computers, there is no doubt that that invention would be patentable.[18]

Computer programs are, according to the foregoing definition, software and they also have been defined as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result,"[19] that is, in order to process information

---

[10] And sometimes human beings.

[11] *I.e.*, data. For further discussion of the nature of computers, *see infra* Part VI.A.

[12] For a discussion of the nature of information and data—which is information that is interesting enough for someone to process it in a computer—*See infra* Part VI.C.

[13] The term "software" is misleading to the extent that it suggests that software is—that computer programs are—the same sort of "ware" as the hardware, *i.e.*, the computer, upon which it runs. *See infra* Part VI.B.

[14] In particular, they are not—and thus they are not patentable as—machines, manufactures, or compositions of matter. *See infra* note 20.

[15] For a quick definition of data, *see infra* note 17.

[16] At least in those cases where they are not human beings. *See infra* text accompanying notes 23 and 24.

[17] Or, slightly more precisely, data. "Data" is simply information that someone finds interesting and perhaps worth processing. As to what information is, *see infra* Part VI.C.

[18] As a machine. *See infra* note 20.

[19] This definition comes from the Copyright Act, 17 U.S.C. § 101. One caution about this definition: the term "instructions" does not imply that computers understand and obey instructions the way human beings—or even dogs—do. Digital computers mechanically carry out their instructions in the same manner that an automobile obeys the instructions transmitted to it by the steering wheel or the brake, quite without any understanding. Human computers, on the other hand, may, but are not required to, understand what it is that they are doing. The Copyright Act also provides that: "In no case does copyright protection . . . extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied . . . ." *Id.* at § 101b. That,

is some specified way. Now, since the Patent Act says that processes are patentable,[20] it might seem that the Court's holding in *Benson* was clearly wrong. But the Court in *Benson*, of course, was perfectly well aware that processes are patentable, and yet it still held that means of processing data, even novel and useful ones, specified by an algorithm, which is to say any computer program,[21] cannot be patented, *because they are the equivalent of mental processes that can be carried out not only by machines, but also by critters like us using nothing but "head and hand."*[22] There once was, after all, a time within the memory of some who are still alive[23] when "computer" was a job description and "computers" were people.[24]

There are lots of processes that involve the processing of information that no one would dream[25] could be patented: hearing, reading, thinking, speaking, writing, printing, and publishing, for example, processes that[26] are protected by the Freedoms of Speech

---

of course, raises the question of whether software can be copyrighted, a question that is not further discussed in this article.

[20] Part 101 of the Patent Act provides that:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title. 35 U.S.C. § 101 and Part 100(b) defines "process" as "process, art or method, and includes a new use of a known process, machine, manufacture, composition of matter, or material."

35 U.S.C. § 100(b).

[21] Every computer program is an algorithm, ultimately composed of simple steps that can be executed with stupendous reliability by one simple mechanism or another. Electronic circuits are the usual choice, but the power of computers owes nothing (save speed) to the causal peculiarities of electrons darting about on silicon chips. The very same algorithms can be performed (even faster) by devices shunting photons in glass fibers, or (much, much slower) by teams of people using paper and pencils. DANIEL C. DENNET, DARWIN'S DANGEROUS IDEA: EVOLUTION AND MEANINGS OF LIFE 50 (Simon & Schuster Inc. 1995). *See infra* text accompanying note 443.

[22] "A digital computer . . . operates on data expressed in digits, solving a problem by doing arithmetic as a person would do it by head and hand." 409 U.S. at 65. *See infra* note 76 Other reasons given by the Court in *Benson* for the unpatentability of algorithms is that they are unpatentable ideas and that they are—or are the equivalent of—natural laws that express universal truths that cannot be invented, but only discovered. *See infra* note 40 and accompanying text.

[23] Like me.

[24] *See infra* Part VI.A.1. I even worked as a computer for a couple of evenings in the mid-fifties of the last century when I was employed as a gravity observer up in the Northwest Territories of Canada. It should be noted that people are kickable and tangible just like other computers, even if they do not make very good doorstops. *See supra* text accompanying note 17. It should also be noted that human computers were not required to understand what it was that they were doing. *See supra* note 19 and *infra* note 62.

[25] Except perhaps for some patent attorney dreaming of becoming the master of the world.

[26] With the possible exception of thinking.

and of the Press of the First Amendment.[27] I doubt that even the most determined advocates of software patents would argue that "mental processes" as such are, or should be, patentable.

II. YOU CAN'T PATENT SOFTWARE: *BENSON* AND ITS PROGENY

There are three cases decided by the Supreme Court that collectively hold beyond any doubt that software as such cannot be patented:[28] *Gottschalk v. Benson*,[29] *Parker v. Flook*,[30] and *Diamond v. Diehr*.[31]

A. Gottschalk v. Benson

Benson was the first case in which the Supreme Court considered the issue of whether software is patentable.[32] Here is how Justice Douglas, who wrote the opinion for the unanimous[33] Court in *Benson*, described the patent claims that were at issue in that case:

> Respondents filed in the Patent Office an application for an invention which was described as being related "to the processing of data by program[34] and more particularly to the programmed conversion of numerical information" in general-purpose digital computers.[35] They claimed a method for converting binary-coded decimal (BCD) numerals into pure binary numerals.[36] The claims were not limited to any particular art or technology, to any particular apparatus or machinery, or to any particular end use. They purported to cover any use of the claimed method in a general-purpose digital computer of any type. . . .
>
> . . . .

---

[27] *See infra* Part VI.B.5.

[28] For an excellent short description of what these cases hold, as well as of the constitutional authority to grant patents, see the passage from the dissent of Chief Judge Archer in the *Alappat* case that is quoted *infra* Part III.A.1.

[29] 409 U.S. 63 (1972).

[30] 437 U.S. 584 (1978).

[31] 450 U.S. 175 (1981).

[32] The patent claim in issue was filed by Gary *Benson* and Arthur Talbott, employees of Bell Telephone Laboratories to whom they had assigned their patent rights. *See* Donald S. Chisum, *The Patentability of Algorithms*, 47 U. PITT. L. REV. 959, 972, 972 n.45 (1986).

[33] Justices Stewart, Blackmun, and Powell did not take part in the consideration of the case. *See, e.g., id.* at 977 n.61.

[34] As we shall see, all that programmable computers do is "the processing of data by program." *See infra* Part VI.A.

[35] All that general purpose digital computers can do is convert—*i.e.*, process—information that can be represented by numerals.

[36] *See infra* text accompanying note 52.

> The patent sought is on a method of programming a general-purpose digital computer to convert signals from binary-coded decimal form into pure binary form. A procedure for solving a given type of mathematical problem is known as an "algorithm."[37] The procedures set forth in the present claims are of that kind; that is to say, they are a generalized formulation for programs to solve mathematical problems of converting one form of numerical representation to another. From the generic formulation, programs may be developed as specific applications.[38]

The question before the Court was whether the method described and claimed—whether the claimed way of converting numerical information—was a patentable "process."

Even though the Patent Act defines a method as a process,[39] the Court in *Benson* held that the claimed method was not patentable, because, in the first place, a mathematical process is an "idea" and ideas themselves are not patentable and, in the second place, mathematical processes represent fundamental truths or laws of nature and such truths or laws are not patentable.[40] The Court, of course, did not deny that processes can be patented, but it said that, "Transformation and reduction of an article 'to a different state or thing' is the clue to the patentability of a process claim that does not include particular machines."[41]

The Court in *Benson* held that the process sought to be patented did not meet this test, which amounts to a holding that a means of processing data or information[42] is not a patentable process. And, since one of the claims in *Benson* did include a specific reference to a particular type of machine, a so-called "reentrant shift register,"[43] and both claims could only be used in conjunction with a particular type of machine—a digital computer[44]—it also amounts to a holding that

---

[37] As we shall see, all computer programs are algorithms, although not all algorithms are computer programs. *See infra* Part VI.B.1.

[38] 409 US at 64–65.

[39] *See supra* note 20.

[40] These two reasons are not separate, for the truths or laws of mathematics are both a special case of the truths or laws of nature and a classic example of what is meant by an idea. Thus, for example, the idea that 2+2 = 4 is a truth or law of mathematics. (Using the normal definitions of "2" and "+").

[41] 409 U.S. at 70. In the context of *Benson* it is clear that the Court did not consider the data that was processed was the type of "article" to which this description applies.

[42] As to the meaning of "data" and "information" *See infra* Part VI.C.

[43] 409 U.S. at 73. For more on registers, *see infra* Part VI.A.6.

[44] *See infra* text accompanying notes 68–70.

such a method is not patentable even if—or especially if—it requires the use of a computer.[45]

1. Proceedings Below

Initially the patent examiner and the Patent Board of Appeals in the Patent Office both held that the processes described in *Benson's* claims[46] were not patentable because they "set forth 'mental

---

[45] Or a component part thereof like a shift register. On the other hand, nothing in *Benson* suggests that a new type of computer or shift register could not be patented like any other type of machine.

[46] The two independent claims at issue in *Benson* are set out in an appendix to the opinion of the Supreme Court in that case. 409 U.S. at 73.

Claim 8 reads:

"The method of converting signals from binary coded decimal form into binary which comprises the steps of

(1) storing the binary coded decimal signals in a reentrant shift register,
(2) shifting the signals to the right by at least three places, until there is a binary '1' in the second position of said register,
(3) masking out said binary '1' in said second position of said register,
(4) adding a binary '1' to the first position of said register,
(5) shifting the signals to the left by two positions,
(6) adding a '1' to said first position, and,
(7) shifting the signals to the right by at least three positions in preparation for a succeeding binary '1' in the second position of said register."

Claim 13 reads:

"A data processing method for converting binary coded decimal number representations into binary number representations comprising the steps of

(1) testing each binary digit position '1,' beginning with the least significant binary digit position, of the most significant decimal digit representation for a binary '0' or a binary '1';
(2) if a binary '0' is detected, repeating step (1) for the next least significant binary digit position of said most significant decimal digit representation;
(3) if a binary '1' is detected, adding a binary '1' at the (i + 1)th and (i + 3)th least significant binary digit positions of the next lesser significant decimal digit representation, and repeating step (1) for the next least significant binary digit position of said most significant decimal digit representation;
(4) upon exhausting the binary digit positions of said most significant decimal digit representation, repeating steps (1) through (3) for the next lesser significant decimal digit representation as modified by the previous execution of steps (1) through (3); and,
(5) repeating steps (1) through (4) until the second least significant decimal digit representation has been so processed."

processes' and 'mathematical steps'," neither of which were patentable subject matter "as construed by a long line of decisions."[47]

The Court of Customs and Patent Appeals ("C.C.P.A") rejected this analysis because, even though in theory the process disclosed in the claims could be carried out by a human being using pencil and paper, the only practical application of the claims involved the use of a machine—a general purpose digital computer—and general purpose digital computers do not perform mental steps.[48]

Before reaching that conclusion, the Court of Customs and Patent Appeals explained what the claimed method or process was all about, in a passage that you may feel tells you more about bits and the representations of numbers than you ever wanted to know:[49]

> Most digital computers perform their computing operations on information[50] in binary form, a system of representation having only two elementary constituents, called "bits," indicated by "1" and "0". Men, on the other hand, are accustomed to quantitative information in decimal form and, for the computer to work on or utilize it, it must be converted into binary form. It has become a general practice to make this conversion in two stages: from decimal to BCD; from BCD to binary. The following table shows the ten familiar decimal digits and their binary equivalents expressed in groups of four bits:

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110

---

[47] *In re Benson*, 58 C.C.P.A. 1134, 1137 (C.C.P.A. 1971). For a short history of the "mental steps" doctrine. *See infra* text accompanying note 100.

[48] *Id.* That, I submit, amounted to saying that if one uses a tool like a computer or an abacus or one's fingers to carry out a calculation that calculation is not a mental process.

[49] This description does, however, serve as a pretty good introduction to the nature of computer programs down at the lowest level where the instructions interact directly with the hardware.

[50] *Nota bene* this recognition that computers perform their operations upon "information"—that computers process information. For a discussion of the nature of information and an explanation of why methods of processing information should not be patentable, *see infra* Part VI.C.

7	0111
8	1000
9	1001

> In BCD notation the decimal number 53 would be represented as 0101 0011, the binary form of 5 followed by the binary form of 3, each group of four bits being a BCD "digit." Although the BCD notation is in terms of "1" and "0" only and can be represented in a binary machine, it is not in a form in which the machine can utilize it to perform its computing operations. In true binary the decimal number 53 is represented by 110101. The problem is to convert the intermediate BCD representation into the true binary. Various ways of doing this were known prior to appellants' invention. They claim to have discovered a better and simpler way of doing it having various advantages recited in the specification such as reducing the number of steps required to be taken, dispensing with the repetitive storing and retrieval of partially converted information, eliminating the need for interchanging signals among various equipment components and the need for auxiliary equipment, and decreasing the chance of error.[51]

It is, I think, important to realize that the Court of Customs and Patent Appeals did not reach its decision that *Benson*'s claims describing a process for converting BDC representations of numbers to pure binary representations were patentable on any theory that mental processes and mathematical steps are patentable; rather it held that, since the steps and processes described in the claim were to be carried out by a computer they were not mental processes or mathematical steps, a claim that clearly was rejected by the Supreme Court in *Benson*.

Here is what the C.C.P.A. had to say about Claim 8[52]—the claim describing the use of a "reentrant shift register":

> We have set forth claim 8 above and have stated that the examiner and board rejected it because they considered it to be directed to "mental processes" and "mathematical steps." . . . [T]he question we have here [is]: ". . . would a *reasonable* interpretation of the claims include coverage of the process implemented by the human mind?" The answer clearly is

---

[51] 58 C.C.P.A. at 1137.

[52] *See supra* note 46.

> "No." . . . .[53] Claim 8 is for a method to be practiced in part on particular apparatus specified to be a "reentrant shift register." . . . . At argument, the Patent Office Solicitor admitted that the reference to this piece of apparatus in the claim was, for him, its "most embarrassing phrase."[54] It is not only a phrase; it is referred to expressly in elements (1), (2), (3), (4) and (7) and by implication in element (5) which refers to "shifting." Claim 8, moreover, refers to the operations of storing, shifting, and masking "signals" which, by a reasonable interpretation in the light of the specification, can only mean signals of the kind upon which the disclosed electronic digital computer hardware operates. . . . The process can be carried out with no intervention by a human being once the apparatus is set up—that is to say, the appropriate computer system has been assembled and programmed. . . .

On the other hand, the process can only be carried out in order to satisfy human ends, directly or indirectly, and the ultimate outcome of the process is useful only if there is a human being to interpret it. The fact that the process can be carried out with no intervention by a human being after the computer has been programmed is not a very persuasive argument for saying that the program itself is not an idea of a human mind.

The court also rejected the argument of the Patent Office Solicitor that the claimed method was not a patentable process because it is merely " a "tool of the mind,"[55] and that the method was basically "mental" in character, because its "work stuff" was numbers which are mathematical abstractions, supporting this rejection with the non sequitur that "Cash registers, bookkeeping machines, and adding machines also work only with numbers[56] but this has never been

---

[53] Justice Douglas's opinion in *Benson* in effect says that the answer clearly is "Yes."

[54] 58 C.C.P.A. at 1142. There was, as far as I can see, no reason for the Solicitor to be embarrassed by the fact that claim 8 specified that the described program would run on a particular apparatus. That fact certainly did not embarrass Justice Douglas when the issue reached the Supreme Court. The supposed embarrassment, and the confusion, apparently arose from the fact that it sounds strange to say that the computations done by a computer are mental processes, rather than saying that they are "steps in the performance of mental processes" or something like that. Justice Douglas avoided this embarrassment by not referring directly to mental processes, but rather to "ideas" and "truths or laws of nature."

[55] If a computer is not a "tool of the mind," then what is it?

[56] The fact that cash registers, bookkeeping machines, and adding machines—and computers and their components–are patentable as the machines that they are does not in any way suggest that the instructions on how to use those machines are themselves machines.

considered a ground for taking them out of the 'machine' category of section 101."[57]

And here is what the court had to say about the other independent claim:

> [Claim 13] was rejected on the same ground as claim 8 from which it differs in containing no reference to any apparatus and in referring to the thing operated upon not as "signals" but as "representations." The claimed method is one for converting "binary coded decimal number representations" into "binary number representations."[58] The supporting disclosure against which the claim must be reasonably interpreted is the identical programmed digital computer system which supports claim 8. The operational steps to be carried out call for "testing each binary digit position" to determine whether it is a "binary '0' "or a "binary '1'" and performing a specified act according to what is found, *i.e.*, moving to the next position if it is "0" and repeating the test or adding a binary "1" at two specified positions and then moving to the next position, and so on, each action being prescribed so that no human judgment or decision is required—merely observation and taking prescribed action according to what is observed. . . .[59]

Now, as I look back on it, I recall that, during the time when I was a computer[60] I only had to make observations and take prescribed actions[61] although, being human, I saw incapable of totally forgoing the making of human judgments and decisions.[62]

The court then proceeded to describe the "apparatus" that it claimed was disclosed in that claim:

> Apparatus, machinery, "hardware"—whatever it may be called—is disclosed by which the steps *can be* carried out without human intervention but at the same time, since the claim does not itself call for any particular hardware, the method within the claim can be practiced either with

---

[57] 58 C.C.P.A. at 1142.

[58] What exactly is a "representation"?

[59] 58 C.C.P.A. at 1142.

[60] *See supra* note 24.

[61] I am afraid that I had very little understanding about what I was doing as I pushed back and forth the slides on the slide rule in the way that I had been instructed to do.

[62] The judgments and decisions that I did make almost certainly led to errors that reduced the accuracy of my computations.

> apparatus other than that described or with the simplest of equipment which will enable one to provide and to manipulate "binary coded decimal representations" and convert them into "binary number representations." This could *in theory* be any kind of writing implement and any kind of recording medium—"pencil and paper"—or even, we suppose, red and blue poker chips and a surface to put them on or slots to put them in so that "0"s and "1"s can be represented. . . .[63]

This is a remarkable, and important, concession, for it appears to equate computers with pencils and paper and seems to amount to the claim that any, except the most trivial, mathematical operations are patentable. If pencil and paper counts as an apparatus that does not involve mental processes, then a formula for finding the value of x when $ax^2 + bx + c = 0$, or any other formula of similar or greater complexity would be patentable; in other words one could, were it novel, get a patent on the quadratic formula,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

provided only that the inventor wrote it down using pencil and paper or some equivalent technology.

And finally the court argued:

> The only argument put forward by the Patent Office for holding claim 13 non-statutory under section 101 is that the method is basically "mental" in character. Looking at the present case in the light of all its circumstances, we observe in claim 13 a process consisting of a sequence of steps which can be carried out by machine implementation . . . . In no case is the exercise of judgment required or even the making of a decision as between alternatives.
>
> Realistically, the process of claim 13 has no practical use other than the more effective operation and utilization of a *machine* known as a digital computer.[64] It seems beyond question that the machines—the computers—are in the

[63] 58 C.C.P.A. at 1142–43.

[64] It should be noted that Justice Douglas used this same fact as an argument for holding *Benson*'s discovery to be patentable.

> technological field, are a part of one of our best-known technologies, and are in the "useful arts" rather than the "liberal arts," as are all other types of "business machines," regardless of the uses to which their users may put them. How can it be said that a process having no practical value other than enhancing the internal operation of those machines is not likewise in the technological or useful arts? . . . [65]

Of course, the fact that computer hardware lies within the traditional technological arts does not in anyway suggest that computer software—such as the quadratic equation—or logical operations like those involved in *Benson*—are not in the liberal arts that have never been protected by patents. Still, logically or not, the C.C.P.A. concluded that "the Patent Office has put forth no sound reason why the claims in this case should be held to be non-statutory."[66] And reversed the decision of the Patent Office.

Now, at first glance, this may appear to have been a reasonable decision, since computers—not having minds—quite obviously do not indulge in mental processes or take mental steps. But this overlooks the critical fact that people—who do have minds—use computers as an aid to thinking—an aid to carrying out mental processes—and as an aid to carrying out mathematical steps.[67]

It is thus hardly surprising that the Supreme Court reversed the decision of the C.C.P.A.

2. The Decision of the Supreme Court

The Supreme Court—which has, of course, the final word in such matters—rejected the holding and the reasoning of the Court of Custom and Patent Appeals, saying:

> It is conceded that one may not patent an idea.[68] But in practical effect that would be the result if the formula for converting BCD numerals to pure binary numerals were patented in this case. The mathematical formula[69] involved

---

[65] 58 C.C.P.A. at 1143.

[66] *Id.* at 688.

[67] Thus, for example, in my old age I find that I am incapable of carrying out all but the most simple mathematical steps without the assistance of a computer or a calculator nor would I have been able to carry out the mental steps that were necessarily involved in writing this article without the assistance of computers and their software.

[68] Note that the term "idea" is used here as the equivalent of a "mental process" or, perhaps, in the Platonic sense of an ultimate reality—or both.

[69] Notice that Justice Douglas seems to use the terms "algorithm" and "mathematical formula" almost interchangeably.

> here has no substantial practical application except in connection with a digital computer, which means that if the judgment below is affirmed, the patent would wholly preempt the mathematical formula and in practical effect would be a patent on the algorithm itself.[70]

Justice Douglas, who never bothered to mention the lower court's rejection of the mental steps doctrine,[71] here clearly rejected the argument that the algorithm was patentable because it was carried out by a machine—by a general purpose digital computer—and instead said that the "mathematical formula" was unpatentable precisely because it had "no substantial practical application except in connection with a digital computer." This is, I submit, equivalent to saying that that the algorithm was unpatentable because its only practical application was in the processing of "information" or "data,"[72] which, of course, is all that computers do.

3. The Issues in Benson *in More Detail*

The Court in *Benson* stressed the fact that digital computers are used only to process digits[73]—a claim that is almost tautological and, at the same time, I suspect, almost incomprehensible to one whose only experience with computers involves "clicking" on an "icon" or sending email[74] or who thinks that the term "digits" only refers to fingers.

---

[70] *Gottschalk v. Benson*, 409 U.S. 63, 71–72 (1972). Immediately before this passage, the Court said:

> It is argued that a process patent must either be tied to a particular machine or apparatus or must operate to change articles or materials to a "different state or thing." We do not hold that no process patent could ever qualify if it did not meet the requirements of our prior precedents. It is said that the decision precludes a patent for any program servicing a computer. We do not so hold. It is said that we have before us a program for a digital computer but extend our holding to programs for analog computers. We have, however, made clear from the start that we deal with a program only for digital computers. It is said we freeze process patents to old technologies, leaving no room for the revelations of the new, onrushing technology. Such is not our purpose. . . . The fact that the Court was careful not to extend its holding beyond the issues raised in *Benson* does not, of course, mean that the Court held that some of those processes are, in fact, patentable.

[71] The mental steps doctrine was, however, later favorably referred to by the dissenters in *Diehr*. *See infra* text accompanying note 100.

[72] With the qualification, that some may find significant, that the results would only be useful in other computer programs. *See infra* Part VI.C.

[73] That is to say "numerals."

[74] Anyone who believes that knowing how to click and to send email makes one "computer literate" should read: Neil Stephenson, *In the Beginning Was the Command Line* (1999), http://www.cryptonomicon.com/beginning.html.

Describing the claims at issue, the Justice Douglas said:

> A digital computer . . . operates on data expressed in digits,[75] solving a problem by doing arithmetic as a person would do it by head and hand.[76] Some of the digits are stored as components of the computer. Others are introduced into the computer in a form which it is designed to recognize.[77] The computer operates then upon both new and previously stored data. The general-purpose computer is designed to perform operations under many different programs.
>
> . . . .
>
> The patent sought is on a method of programming a general-purpose digital computer to convert signals[78] from binary-coded decimal form into pure binary form. A procedure for solving a given type of mathematical problem is known as an "algorithm." The procedures set forth in the present claims are of that kind; that is to say, they are a generalized formulation for programs to solve mathematical problems of converting one form of numerical representation to another.[79] From the generic formulation, programs may be developed as specific applications.[80]
>
> The decimal system uses as digits the 10 symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The value represented by any digit depends, as it does in any positional system of notation, both on its individual value and on its relative position in the numeral. Decimal numerals are written by placing digits in the appropriate positions or columns of the numerical sequence, *i.e.*, "unit" (10^0), "tens" (10^1), "hundreds" (10^2), "thousands" (10^3), etc. Accordingly, the numeral 1492 signifies $(1 \times 10^3) + (4 \times 10^2) + (9 \times 10^1) + (2 \times 10^0)$.

---

[75] That is the fundamental fact about digital computers: they operate on—*i.e.*, process—data, *i.e.*, information, represented as digits.

[76] note that the person who does the computations "by head and hand" may not have any idea of why the computations are being done or what they represent. *See infra* Part VI.A.1.

[77] The word "recognize" is not really felicitous here, for computers, of course, do not recognize things the way people and other sentient beings recognize them.

[78] *I.e.*, data received as part of a temporal process, in contrast to "signs" which represent data stored in some tangible medium of expression.

[79] The "numerical representations" are the way that the data are represented as numbers.

[80] *Nota bene* the relation between algorithms and programs: the algorithms are general descriptions such as one finds in patent claims; the actual programs, on the other hand, are implementations of the algorithm that can be run on a particular type of digital computer or written in a particular computer language.

The pure binary system of positional notation uses two symbols as digits – 0 and 1, placed in a numerical sequence with values based on consecutively ascending powers of 2. In pure binary notation, what would be the tens position is the twos position; what would be hundreds position is the fours position; what would be the thousands position is the eights. Any decimal number from 0 to 10 can be represented in the binary system with four digits or positions as indicated in the following table.

Shown as the sum of powers of 2

Decimal		2^3 (8)		2^2 (4)		2^1 (2)		2^0 (1)		Pure Binary
0	=	0	+	0	+	0	+	0	=	0000
1	=	0	+	0	+	0	+	2^0	=	0001
2	=	0	+	0	+	2^1	+	0	=	0010
3	=	0	+	0	+	2^1	+	2^0	=	0011
4	=	0	+	2^2	+	0	+	0	=	0100
5	=	0	+	2^2	+	0	+	2^0	=	0101
6	=	0	+	2^2	+	2^1	+	0	=	0110
7	=	0	+	2^2	+	2^1	+	2^0	=	0111
8	=	2^3	+	0	+	0	+	0	=	1000
9	=	2^3	+	0	+	0	+	2^0	=	1001
10	=	2^3	+	0	+	2^1	+	0	=	1010

The BCD system using decimal numerals replaces the character for each component decimal digit in the decimal numeral with the corresponding four-digit binary numeral, shown in the right-hand column of the table. Thus decimal 53 is represented as 0101 0011 in BCD, because decimal 5 is equal to binary 0101 and decimal 3 is equivalent to binary 0011. In pure binary notation, however, decimal 53 equals binary 110101. The conversion of BCD numerals to pure binary numerals can be done mentally through use of the foregoing table. The method sought to be patented varies the ordinary arithmetic steps a human would use by changing the order of the steps, changing the symbolism for writing the multiplier used in some steps, and by taking subtotals after each successive operation. The mathematical procedures can

> be carried out in existing computers long in use, no new machinery being necessary. And, as noted, they can also be performed without a computer.[81]

Now you may well feel at this point that Justice Douglas has insisted on telling you more about binary digits than you ever wanted to know, but the explanation was necessary if he was to explain what the claims were and why the claims were not patentable. And I am afraid that we will have to spend more time trying to get our minds around the relation between numbers and numerals and especially the concept of binary digits—or "bits"—because bits and patterns of bits are, when one gets right down to fundamentals, all that are operated on by digital computers.

4. The Critical Holding

Although there can perhaps be debate about the exact reasons for the Court's rejection of the claims in *Benson*, at least one thing is clear: the Court held that algorithms for processing data—in particular instructions to be carried out by a general propose digital computer—are not patentable subject matter. Let me repeat that: in *Benson* the Supreme Court held that mathematical algorithms[82] for processing data to be carried out on a computer are not patentable. And the Supreme Court has never overruled—or even questioned—that holding.

B. Parker v. Flook

Flook[83] did little more than reaffirm the holding in *Benson* and make it clear that no mathematical calculation, even if it is to be performed with the aid of a computer, is patentable. The Court's opinion, by Justice Stevens, begins:

> Respondent applied for a patent on a "Method for Updating Alarm Limits." The only novel feature of the method is a mathematical formula. In *Gottschalk v. Benson* we held that the discovery of a novel and useful mathematical formula may not be patented.[84] The question in this case is whether

---

[81] *Gottschalk*, 409 US at 64–67 (footnote omitted).

[82] Which the Court also referred to as mathematical formulae. *See id.* at 71. As to what an algorithm is, *see infra* Part VI.B.1. It should be noted that all patent claims are—or, at least, should be—algorithms and that every computer program, no matter how trivial, satisfies the definition of a "mathematical algorithm."

[83] 437 U.S. 594.

[84] Of course, if the formula is not novel or not useful it not patentable for those reasons as

> the identification of a limited category of useful, though conventional, post-solution applications of such a formula makes respondent's method eligible for patent protection.[85]

The Court held that it did not.

Perhaps the major difference between *Benson* and *Flook* is that in the latter case there was no discussion of numerals or binary digits, the Court referring only to "numbers," and the algorithm at issue was also described as a "formula." Thus in *Flook* the Court said:

> An "alarm limit" is a number. During catalytic conversion processes, operating conditions such as temperature, pressure, and flow rates are constantly monitored. When any of these "process variables" exceeds a predetermined "alarm limit," an alarm may signal the presence of an abnormal condition indicating either inefficiency or perhaps danger. Fixed alarm limits may be appropriate for a steady operation, but during transient operating situations, such as startup, it may be necessary to "update" the alarm limits periodically.
>
> Respondent's patent application describes a method of updating alarm limits. In essence, the method consists of three steps: an initial step which merely measures the present value of the process variable (e. g., the temperature); an intermediate step which uses an algorithm[86] to calculate an updated alarm-limit value; and a final step in which the actual alarm limit is adjusted to the updated value. The only difference between the conventional methods of changing alarm limits and that described in respondent's application rests in the second step—the mathematical algorithm or formula.[87] *Benson* Using the formula, an operator can calculate an updated alarm limit once he knows the original alarm base, the appropriate margin of safety, the time interval that should elapse between each updating, the current temperature (or other process variable), and the appropriate

---

well.

[85] *Parker*, 437 U.S. at 585 (citation omitted).

[86] We use the word "algorithm" in this case, as we did in *Gottschalk v. Benson* to mean "[a] procedure for solving a given type of mathematical problem . . . ." (footnote by the Court, renumbered) (citation omitted).

[87] Note how the Court here equates formulae with algorithms and treats *Benson* as dealing only with mathematical algorithms.

> weighting factor to be used to average the original alarm base and the current temperature.
>
> The patent application does not purport to explain how to select the appropriate margin of safety, the weighting factor, or any of the other variables. Nor does it purport to contain any disclosure relating to the chemical processes at work, the monitoring of process variables, or the means of setting off an alarm or adjusting an alarm system. All that it provides is a formula for computing an updated alarm limit. Although the computations can be made by pencil and paper calculations, the abstract of disclosure makes it clear that the formula is primarily useful for computerized calculations producing automatic adjustments in alarm settings.
>
> The patent claims cover any use of respondent's formula for updating the value of an alarm limit on any process variable involved in a process comprising the catalytic chemical conversion of hydrocarbons. Since there are numerous processes of that kind in the petrochemical and oil-refining industries, the claims cover a broad range of potential uses of the method. They do not, however, cover every conceivable application of the formula.[88]

In *Flook*, the Court not only followed *Benson* but clarified and extended it, holding that the addition to the claim of "postsolution" activity that limited the possible uses of the algorithm did not make the algorithm a patentable process:

> The line between a patentable "process" and an unpatentable "principle" is not always clear. Both are "[conceptions] of the mind, seen only by [their] effects when being executed or performed." In *Benson* we concluded that the process application in fact sought to patent an idea, noting that
>
> > "[the] mathematical formula involved here has no substantial practical application except in connection with a digital computer, which means that if the judgment below is affirmed, the patent would wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself."

---

[88] *Parker*, 437 U.S. at 585–86 (footnotes omitted.)

> Respondent correctly points out that this language does not apply to his claims. He does not seek to "wholly preempt the mathematical formula," since there are uses of his formula outside the petrochemical and oil-refining industries that remain in the public domain. And he argues that the presence of specific "post-solution" activity—the adjustment of the alarm limit to the figure computed according to the formula—distinguishes this case from *Benson* and makes his process patentable. We cannot agree.
>
> The notion that post-solution activity, no matter how conventional or obvious in itself, can transform an unpatentable principle into a patentable process exalts form over substance. A competent draftsman could attach some form of post-solution activity to almost any mathematical formula; the Pythagorean theorem would not have been patentable, or partially patentable, because a patent application contained a final step indicating that the formula, when solved, could be usefully applied to existing surveying techniques.[89] The concept of patentable subject matter under § 101 is not "like a nose of wax which may be turned and twisted in any direction . . . ."[90]

The Court was, however, careful to point out that: "While a scientific truth, or the mathematical expression of it, is not patentable invention, a novel and useful structure created with the aid of knowledge of scientific truth may be." And then the Court explained that, for a process to be patentable: "The process itself, not merely the mathematical algorithm, must be new and useful. Indeed, the novelty of the mathematical algorithm is not a determining factor at all. Whether the algorithm was in fact known or unknown at the time of the claimed invention, as one of the 'basic tools of scientific and technological work, it is treated as though it were a familiar part of the prior art."[91]

---

[89] It should be noted that in *Benson* there was a specific end use contemplated for the algorithm—utilization of the algorithm in computer programming. Of course, as the Court pointed out, the formula had no other practical application; but it is not entirely clear why a process claim is any more or less patentable because the specific end use contemplated is the only one for which the algorithm has any practical application. (footnote by the Court, renumbered) (citation omitted).

[90] 437 U.S. at 589–90 (citations omitted).

[91] *Id.* at 591–92 (citation omitted). Thus in a case like *Diamond v. Diehr*, *see infra* Part II.C, where the claim covers patentable subject matter but the only novelty is in a mathematical formula—though in *Diehr* itself there appears to have been no novelty whatsoever—the claim would be unpatentable for lack of novelty, no matter how novel the formula.

1. The Dissent in Flook

There was a dissent filed in *Flook* by Justice Stewart in which Chief Justice Berger and Justice Rehnquist joined, but that dissent in no way questioned the holding in *Benson*. Rather, anticipating the majority opinion in *Diamond v. Diehr*,[92] the dissenters argued that the claimed process did satisfy the requirements of Section 101 of the Patent Act, since it involved physical changes, even though it was probably unpatentable under Sections 102 and 103.

Justice Stewart said:

> Section 101 is concerned only with subject-matter patentability. Whether a patent will actually issue depends upon the criteria of §§ 102 and 103, which include novelty and inventiveness, among many others. It may well be that under the criteria of §§ 102 and 103 no patent should issue on the process claimed in this case, because of anticipation, abandonment, obviousness, or for some other reason. But in my view the claimed process clearly meets the standards of subject-matter patentability of § 101.[93]

C. Diamond v. Diehr

The only other case in which the Supreme Court considered the patentability of algorithms or mathematical formulae was *Diamond v. Diehr*.[94] The claim involved in that case was for "a process for molding raw, uncured synthetic rubber into cured precision products. The process uses a mold for precisely shaping the uncured material under heat and pressure and then curing the synthetic rubber in the mold so that the product will retain its shape and be functionally operative after the molding is completed."[95]

Not surprisingly the Court[96] held that the subject matter of the claim was patentable under Section 101 of the Patent Act even though one element of the claim was the use of a computer to solve a mathematical formula. The only question before the Court was whether the claim presented patentable subject matter under Section

---

[92] 450 U.S. 175 (1981).

[93] *Parker*, 437 U.S. at 600 (Stewart, J., dissenting).

[94] 450 U.S. 175 (1981).

[95] 450 U.S. at 177. There was a dissent in *Diehr*, but the dissenters' objection was not to the majority's reasoning, but only to their interpretation of this claim that was at issue. *See infra* Part II.C.1.

[96] In an opinion by Justice Rehnquist in which Justices Brennan, Marshall, and Blackmun joined. *See Diehr*, 450 U.S. at 177.

101, even though it is quite clear—at least it is quite clear to me—that the claim was not novel and was obvious and thus did not satisfy the requirements of Sections 102 and 103 of the Patent Act.[97] The Court in *Diehr* did not question, but rather restated and reaffirmed, the holdings in *Benson* and *Flook*, saying:

> We recognize, of course, that when a claim recites a mathematical formula (or scientific principle or phenomenon of nature), an inquiry must be made into whether the claim is seeking patent protection for that formula in the abstract. A mathematical formula as such is not accorded the protection of our patent laws, *Gottschalk v. Benson*, 409 U.S. 63, 93 (1972), and this principle cannot be circumvented by attempting to limit the use of the formula to a particular technological environment. *Parker v. Flook*, 437 U.S. 584 (1978). Similarly, insignificant post-solution activity will not transform an unpatentable principle into a patentable process. To hold otherwise would allow a competent draftsman to evade the recognized limitations on the type of subject matter eligible for patent protection. On the other hand, when a claim containing a mathematical formula implements or applies that formula in a structure or process which, when considered as a whole, is performing a function which the patent laws were designed to protect (e.g., transforming or reducing an article to a different state or thing), then the claim satisfies the requirements of § 101. Because we do not view respondents' claims as an attempt to patent a mathematical formula, but rather to be drawn to an industrial process for the molding of rubber products, we affirm the judgment of the Court of Customs and Patent Appeals.[98]

Although there have been claims that *Diehr* somehow weakens the presidential value of the holding in *Benson* that algorithms—or mathematical formulae—and computer programs implementing them, are not patentable subject matter, nothing could be further from the

---

[97] The Court specifically stated:

> "In this case, it may later be determined that the respondents' process is not deserving of patent protection because it fails to satisfy the statutory conditions of novelty under § 102 or nonobviousness under § 103. A rejection on either of these grounds does not affect the determination that respondents' claims recited subject matter which was eligible for patent protection under § 101."

Id. at 191.

[98] *Id.* at 191–93.

truth. At the most, *Diehr* allows only that in a few cases in which the claims contain mathematical formula that are well disguised by careful drafting that the claims be treated as covering patentable subject matter under Section 101 of the Patent Act.[99] It should be noted that the leading cases in the Federal Circuit that uphold software patents do not disguise the fact that the claims cover mathematical process that are not tied to a physical result.

1. The Dissent in Diehr

Four justices[100] dissented in *Diehr*, complaining that the Court's decision rested on a misreading of the patent applications in question and "by ignoring the critical distinction between the character of the subject matter that the inventor claims to be novel—the § 101 issue—and the question whether that subject matter is in fact novel—the § 102 issue."[101] The dissent begins with a short but useful history of the "mental steps" doctrine:

> Prior to 1968, well-established principles of patent law probably would have prevented the issuance of a valid patent on almost any conceivable computer program. Under the "mental steps" doctrine, processes involving mental operations were considered unpatentable. The mental-steps doctrine was based upon the familiar principle that a scientific concept or mere idea cannot be the subject of a valid patent. The doctrine was regularly invoked to deny patents to inventions consisting primarily of mathematical formulae or methods of computation. It was also applied against patent claims in which a mental operation or mathematical computation was the sole novel element or inventive contribution; it was clear that patentability could not be predicated upon a mental step. Under the "function of a machine" doctrine, a process which amounted to nothing more than a description of the function of a machine was unpatentable. This doctrine had its origin in several 19th-

---

[99] *See* Dan L. Burk, *Patenting Speech*, 79 TEX. L. REV. 99, 136 (2000):

> Initially, the Supreme Court seemed to indicate in *Gottschalk v. Benson* that software algorithms could not be protected under patent law. The Court reached much the same result in *Parker v. Flook*, but subsequently modified its rule in *Diamond v. Diehr* to hold that software could be patentable under rather stringent constraints. In particular, the Supreme Court specified that the operation of the computer program must be tied to some physical result . . . . (footnotes omitted).

[100] Justices Stevens, Brennan, Marshall, and Blackmun.

[101] *Diehr*, 450 U.S. at 194 (Stevens, J., dissenting).

> century decisions of this Court, and it had been consistently followed thereafter by the lower federal courts. Finally, the definition of "process" announced by this Court in *Cochrane v. Deener*, seemed to indicate that a patentable process must cause a physical transformation in the materials to which the process is applied.[102]

The real disagreement between the majority and the dissenters in *Diehr* did not involve the mental steps doctrine or any other limitation on the patentability of ideas. The only dispute was about the proper reading of the patent claims in question. Thus Justice Stevens, who wrote the dissenting opinion, expressly said:

> As the Court reads the claims in the *Diehr* and *Lutton* patent application, the inventors' discovery is a method of constantly measuring the actual temperature inside a rubber molding press. As I read the claims, their discovery is an improved method of calculating the time that the mold should remain closed during the curing process. If the Court's reading of the claims were correct, I would agree that they disclose patentable subject matter. On the other hand, if the Court accepted my reading, I feel confident that the case would be decided differently.[103]

There thus appears to have been no dispute in *Diehr* about the validity of the rule that an improved method of calculating something is not patentable, the only question was whether the claims in question fell within the ambit of that rule.

The disagreement as to whether lack of obviousness and novelty renders a claim unpatentable under the terms of Section 101[104] does not, however, in any way weaken the holding of the Court in *Diehr* or detract from the fact that the Court in *Diehr* reaffirmed the holdings in *Benson* and *Flook* that:

> A mathematical formula as such is not accorded the protection of our patent laws and this principle cannot be circumvented by attempting to limit the use of the formula to a particular technological environment. Similarly,

---

[102] *Id.* at 195–97 (footnotes and citations omitted).

[103] *Id.* at 206–07 (footnotes omitted).

[104] As opposed to Parts 102 and 103. *See supra* note 92.

> insignificant post-solution activity will not transform an unpatentable principle into a patentable process.[105]

D. Patenting Anything Under the Sun

There is one more decision of the Supreme Court's that should be mentioned here: *Diamond v. Chakrabarty*,[106] a case holding that a new and useful type of bacteria was patentable as a "manufacture" or "composition of matter" and that quotes Congressional committee reports saying that Congress intended patentable subject matter to include "anything under the sun that is made by man."[107]

At first glance this might be taken to mean that anything whatsoever can, if it is useful and novel and made by man, be patented, but the Court in *Chakrabarty* was careful to say:

> This is not to suggest that § 101 has no limits or that it embraces every discovery. The laws of nature, physical phenomena, and abstract ideas have been held not patentable.[108] Thus, a new mineral discovered in the earth or a new plant found in the wild is not patentable subject matter. Likewise, Einstein could not patent his celebrated law that $E = mc^2$; nor could Newton have patented the law of gravity. Such discoveries are "manifestations of . . . nature, free to all men and reserved exclusively to none."[109]

I would only add to this disclaimer mention of the fact that software is composed of text, or of numbers, or of information[110] and that it is used, and can only be used, to instruct a computer to process information or data. When one thinks about it for a moment, unless there is something wrong with the way one thinks, it is obvious that texts,[111] numbers, information, and data are not the sort of things that have a spatial location, are not the sort of things that can be said to be "under the sun."[112]

---

[105] *Diamond*, 450 U.S. at 191–92.
[106] 447 U.S. 303 (1980).
[107] *Id.* at 309.
[108] Citing, *inter alia*, *Benson* and *Flook*.
[109] *Id.* (citations omitted).
[110] *See infra* Part VI.B.
[111] As opposed to physical embodiments of a text.
[112] Or over the sun—or East or West of it for that matter.

III. HAS THE FEDERAL CIRCUIT OVERRULED *BENSON*, *FLOOK* AND *DIEHR*?

In the twenty-five years since the decision of the Supreme Court in *Diehr* the Court has not considered a single case in which there was a challenge to a software patent. Instead, until very recently, it has left issues of patent law to the Federal Court of Appeals for the Federal Circuit, a specialized Article III Court created in 1982 that now hears all appeals from the Board of Patent Appeals in patent cases as well as most such appeals from the federal district courts.[113]

Over the years since then, the Federal Circuit has tried to wriggle its way around the Supreme Court's holdings in *Benson*, *Flook* and *Diehr* and now acts as if it had overruled those decisions.

The initial wriggling took place in a 1994 case named *In re Alappat*;[114] where the Federal Circuit held that a claim that specified that the computations in question were to be run on a machine—that is, a computer—was patentable as a "machine," willfully overlooking the fact that the Supreme Court in *Benson* had held that the claims there were not patentable even though—or perhaps because—they were to be run on a computer. The ultimate wriggle, where the Federal Circuit ended up completely rejecting the holding—and the rationale—of the Supreme Court in *Benson*, occurred in *State Street Bank & Trust Co. v. Signature Financial Group*.[115] In *State Street* the Federal Circuit held that a claim that specified that certain bookkeeping functions were to be calculated on a computer was patentable because those calculations were useful, willfully overlooking the fact that the claims that were held to be unpatentable in *Benson* were also useful.

A. Alappat

In Re *Alappat* involved an appeal from a decision of the Board of Patent Appeals of the Patent Office holding that *Alappat*'s claimed invention of a means of smoothing the curves shown on the screen of

---

[113] Federal Courts Improvement Act of 1982, Pub. L. No. 97-164, 96 Stat. 25 (1982). Before then appeals from decisions of the Patent Office were heard by the Court of Customs and Patent Appeals while appeals from decisions of the federal district courts in patent cases were treated as all other such appeals and were heard by the federal courts of appeal for the numbered circuits. The Supreme Court has recently held that the numbered circuits can hear appeals from the district courts in patent cases where the patent issues were raised, not by the plaintiff's cause of action, but rather by the defendant's counterclaim. *See generally* Holmes Group, Inc. v. Vornado Aircirculation Systems, Inc., 535 U.S. 826 (2002). For further discussion of this development see *infra* Part III.C.

[114] 33 F.3d 1526 (Fed. Cir. 1994).

[115] 149 F.3d 1368 (Fed. Cir. 1998).

a digital oscilloscope was not patentable under the provisions of Section 101 of the Patent Act,[116] because it consisted of nothing but a process consisting of a "mathematical algorithm for computing . . . information."[117]

The Federal Circuit, in a majority opinion written by Circuit Judge Rich, held that "the appealed decision should be reversed because the appealed claims are directed to a 'machine' which is one of the categories named in 35 U.S.C. § 101."[118]

Judge Rich argued:

> [T]he proper inquiry in dealing with the so called mathematical subject matter exception to § 101 alleged herein is to see whether the claimed subject matter as a whole is a disembodied mathematical concept, whether categorized as a mathematical formula, mathematical equation, mathematical algorithm, or the like, which in essence represents nothing more than a "law of nature," "natural phenomenon," or "abstract idea." If so, *Diehr* precludes the patenting of that subject matter. That is not the case here.
>
> Although many, or arguably even all, of the means elements recited in claim 15 represent circuitry elements that perform mathematical calculations, which is essentially true of all digital electrical circuits, the claimed invention as a whole is directed to a combination of interrelated elements which combine to form a machine for converting discrete waveform data samples into anti-aliased pixel illumination intensity data to be displayed on a display means. This is not a disembodied mathematical concept which may be characterized as an "abstract idea," but rather a specific machine to produce a useful, concrete, and tangible result.
>
> . . . .
>
> Furthermore, the claim preamble's recitation that the subject matter for which *Alappat* seeks patent protection is a rasterizer for creating a smooth waveform is not a mere field-of-use label having no significance. Indeed, the preamble specifically recites that the claimed rasterizer converts

---

[116] *See supra* note 20.

[117] *Alappat*, 33 F.3d at 1538.

[118] *Id.* at 1536. Note that this amounts to claiming that, although a process of solving an algorithm is not patentable under Part 101, a machine that does nothing but implement that process is patentable.

> waveform data into output illumination data for a display, and the means elements recited in the body of the claim make reference not only to the inputted waveform data recited in the preamble but also to the output illumination data also recited in the preamble. Claim 15 thus defines a combination of elements constituting a machine for producing an anti-aliased waveform.
>
> The . . . Board majority also erred in its reasoning that claim 15 is unpatentable merely because it "reads on a general purpose digital computer 'means' to perform the various steps under program control." . . . . *Alappat* admits that claim 15 would read on a general purpose computer programmed to carry out the claimed invention, but argues that this alone also does not justify holding claim 15 unpatentable as directed to nonstatutory subject matter. We agree. We have held that such programming creates a new machine, because a general purpose computer in effect becomes a special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software.[119]

This amounts to claiming that an unpatentable mathematical process for converting one form of data to another form of data, becomes a patentable machine if it is claimed that the process can be carried out on a properly programmed general purpose computer. But if that were the law, the claims in *Benson* would have been patentable because they could be carried out on a general purpose computer and in particular a computer containing a shift register.[120]

1. The Dissent in Alappat

Perhaps the most useful antidote to the illogic espoused by the majority in *Alappat* is the dissent of Chief Judge Archer in that case. Since it contains a short and excellent description of the holdings in *Benson*, *Flook*, and *Diehr*, I feel justified in quoting from it at some length.

Now Judge Archer's dissent itself starts with a lengthy quotation that I also consider worthy of quotation:

---

[119] *Id.* at 1544–45 (footnotes omitted).

[120] In *Benson*, the Court said: "The patent sought is on a method of programming a general purpose digital computer to convert signals from binary-coded decimal form into pure binary form." *Gottschalk*, 409 U.S. at 65.

In 1873, George Curtis made certain general observations about patent law, the scope of patentable subject matter being at its heart. He stated them with such force and eloquence, and in my view they have such relevance to the issue we face today, that I repeat them as follows:

> It is necessary . . . to have clear and correct notions of the true scope of a patent right . . . which may be found to assist, in particular cases, the solution of the question, whether a particular invention or discovery is by law a patentable subject.
>
> In this inquiry it is necessary to commence with the process of exclusion; for although, in their widest acceptation, the terms "invention" and "discovery" include the whole vast variety of objects on which the human intellect may be exercised, so that in poetry, in painting, in music, in astronomy, in metaphysics, and in every department of human thought, men constantly invent or discover, in the highest and the strictest sense, their inventions and discoveries in these departments are not the subjects of the patent law. . . . The patent law relates to a great and comprehensive class of discoveries and inventions of some new and useful effect or result in matter, not referable to the department of the fine arts. The matter of which our globe is composed is the material upon which the creative and inventive faculties of man are exercised, in the production of whatever ministers to his convenience or his wants. Over the existence of matter itself he has no control. . . .
>
> The direct control of man over matter consists, therefore, in placing its particles in new relations. This is all that is actually done, or that can be done, namely, to cause the particles of matter existing in the universe to change their former places, by moving them, by muscular power or some other force. But as soon as they are brought into new relations, it is at once perceived that there are vast latent forces in nature, which come to the aid of man, and enable him to produce effects and results of a wholly new

character, far beyond the mere fact of placing the particles in new positions. He moves certain particles of matter into a new juxtaposition, and the chemical agencies and affinities called into action by this new contact produce a substance possessed of new properties and powers, to which has been given the name of gunpowder. He takes a stalk of flax from the ground, splits it into a great number of filaments, twists them together, and laying numbers of the threads thus formed across each other, forms a cloth, which is held together by the tenacity or force of cohesion in the particles, which nature brings to his aid. He moves into new positions and relations certain particles of wood and iron, in various forms, and produces a complicated machine, by which he is able to accomplish a certain purpose, only because the properties of cohesion and the force of gravitation cause it to adhere together and enable the different parts to operate upon each other and to transmit the forces applied to them, according to the laws of motion. It is evident, therefore, that the whole of the act of invention, in the department of useful arts, embraces more than the new arrangement of particles of matter in new relations. The purpose of such new arrangements is to produce some new effect or result, by calling into activity some latent law, or force, or property, by means of which, in a new application, the new effect or result may be accomplished. In every form in which matter is used, in every production of the ingenuity of man, he relies upon the laws of nature and the properties of matter, and seeks for new effects and results through their agency and aid. Merely inert matter alone is not the sole material with which he works. Nature supplies powers, and forces, and active properties, as well as the particles of matter, and these powers, forces, and properties are constantly the subjects of study, inquiry, and experiment, with a view to the production of some new effect or result in matter.

Any definition or description, therefore, of the act of invention, which excludes the application of the

> natural law, or power, or property of matter, on which the inventor has relied for the production of a new effect, and the object of such application, *and confines it to the precise arrangement of the particles of matter which he may have brought together, must be erroneous.*[121]

Chief Judge Archer then explained:

> The Patent Clause of the Constitution empowers the Congress to "promote the Progress of . . . useful Arts, by securing for limited Times to . . . Inventors the exclusive right to their . . . Discoveries."
>
> Congress has implemented this limited grant of power in 35 U.S.C. § 101 by enumerating certain subject matter, the invention or discovery of which may entitle one to a patent: "Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title." The terms used in § 101 have been used for over two hundred years–since the beginnings of American patent law–to define the extent of the subject matter of patentable invention.
>
> Coexistent with the usage of these terms has been the rule that a person cannot obtain a patent for the discovery of an abstract idea, principle or force, law of nature, or natural phenomenon, but rather must invent or discover a practical "application" to a useful end.[122] Thus patent law rewards persons for inventing technologically useful applications, instead of for philosophizing unapplied research and theory. . .[123]

---

[121] *Alappat*, 33 F.3d at 1551–52 (citing G. CURTIS, A TREATISE ON THE LAW OF PATENTS FOR USEFUL INVENTIONS xxiii–xxv (4th ed. 1873) (emphasis added)).

[122] *Id.* (citing, *inter alia*, *Diehr*).

[123] *Id.* at 1552. Chief Judge Archer went on to give this telling example of what would happen were the law otherwise:

He then discussed *Benson*, *Flook*, and *Diehr*:

> The trilogy of Supreme Court cases in this area must be applied to determine whether an invention or discovery in the field of digital electronic related subject matter is within the scope of the patent law. These cases govern both product and process claims.
>
> In the first case, *Gottschalk v. Benson*, the Supreme Court held that claims to a method of converting binary-coded decimal numbers into pure decimal numbers did not recite an invention or discovery within § 101, and thus were ineligible for patent protection. In *Benson*, the claimed method was to be performed specifically in a general purpose digital computer, and one of the claims (claim 8) contained express digital electronic structure limitations by reciting "signals" and a "reentrant shift register." The Court found that the "practical effect" of a patent for the method would be the impermissible award of a

---

Consider for example the discovery or creation of music, a new song. Music of course is not patentable subject matter; a composer cannot obtain exclusive patent rights for the original creation of a musical composition. But now suppose the new melody is recorded on a compact disc. In such case, the particular musical composition will define an arrangement of minute pits in the surface of the compact disc material, and therefore will define its specific structure. Alternatively suppose the music is recorded on the rolls of a player piano or a music box.

Through the expedient of putting his music on known structure, can a composer now claim as his invention the structure of a compact disc or player piano roll containing the melody he discovered and obtain a patent therefor? The answer must be no. The composer admittedly has invented or discovered nothing but music. The discovery of music does not become patentable subject matter simply because there is an arbitrary claim to some structure.

And if a claim to a compact disc or piano roll containing a newly discovered song were regarded as a "manufacture" and within § 101 simply because of the specific physical structure of the compact disc, the "practical effect" would be the granting of a patent for a discovery in music. Where the music is new, the precise structure of the disc or roll would be novel under § 102. Because the patent law cannot examine music for "nonobviousness," the Patent and Trademark Office could not make a showing of obviousness under § 103. The result would well be the award of a patent for the discovery of music. The majority's simplistic approach of looking *only* to whether the claim reads on structure and *ignoring* the claimed invention or discovery for which a patent is sought will result in the awarding of patents for discoveries well beyond the scope of the patent law.

Id. at 1553–54 (citation omitted).

> patent for a discovery in mathematics because the whole of the subject matter sought to be patented was a mathematical formula that had "no substantial practical application except in connection with a digital computer." In *Benson* the Court made clear that it was "dealing with a program only for digital computers."

In the second case, *Parker v. Flook*, the Court held that a claim to a method of updating "alarm limits" (numbers) did not recite an invention or discovery within § 101, and thus was ineligible for patent protection. The claims in *Flook* did not "wholly preempt" the claimed mathematical formula because they did not cover every application of the formula. The claimed method was expressly limited to operation "in a process comprising the catalytic chemical conversion of hydrocarbons," and thereby to application in a particular technological environment. The claimed formula also was "primarily useful for computerized calculations." And the claim recited specific activity beyond the solution of the mathematical formula (so called "post-solution" activity), namely adjusting an "alarm limit" to the figure computed according to the formula. The Court reasoned that the updating of alarm limits in chemical processes was well known, and all that *Flook* purported to invent and claim was a new formula coupled to a computer for doing so (limited to certain postsolution activity in a technological environment). On these facts, the Court reasoned that the claimed invention or discovery was an alleged newly discovered mathematical formula, which was "not the kind of 'discovery' that the statute was enacted to protect."

> In the third case, *Diamond v. Diehr*, the Court held that a process for operating a rubber-molding press was within § 101. An element of the claimed process was a digital computer programmed to perform a mathematical function. It was known that temperature inside a rubber-molding press determined in part the time the press was required to remain closed. The problem faced in the art was that when the press opened during operation, it cooled, thereby changing the amount of time needed for curing. By including a thermocouple or other temperature-detecting device for measuring temperature inside the press, feeding signals to a computer which would repeatedly calculate the cure time and then cause the press to open at the right moment, the applicant claimed to have invented a new, useful, and

> nonobvious precision method of curing rubber. The Court reasoned that the claimed subject matter was, as a whole, a process for precision rubber curing that included a computer performing a mathematical formula; the totality of claimed subject matter was not just the mathematical formula. Therefore, held the Court, the claimed subject matter was eligible for patent protection.
>
> The Court in *Diehr* distinguished its decision in *Flook*. Both cases involved claims including mathematical formulae to be performed by digital electronics, with application in chemical processes. *Flook*'s patent application, however, "did not purport to explain how the variables used in the formula were to be selected." *Flook*'s patent application did not "contain any disclosure relating to the chemical processes at work, the monitoring of process variables, or the means of setting off an alarm system." In contrast, *Diehr*'s claims were neither to the mathematical formula nor to the "the isolated step of 'programming a digital computer.' " They were to a process "beginning with the loading of [a] mold and ending with the opening of [a] press and the production of synthetic rubber product that has been perfectly cured–a result theretofore unknown in the art." The chemical process in *Flook* was not the alleged invention or discovery but only was related tangentially to the mathematical formula; the applicant simply "limited the use of the formula to a particular technological environment" and claimed "insignificant postsolution activity." All this demonstrated that in *Diehr* the applicant was, in substance, asserting and claiming to have invented a new and useful chemical process, thereby qualifying the subject matter for examination under the remaining provisions of the patent law, while in *Flook* as in *Benson* the applicant was, in substance, asserting and claiming as his invention or discovery a mathematical function (to be performed by a computer), thereby placing the subject matter outside the patent law.[124]

Another important point that was made by Chief Judge Archer in *Alappat* was that, if the majority were correct in holding that a specific physical structure, like the holes on a compact disk, is patentable, then a musical work—or any other information—stored

---

[124] *Id.* at 1556–57 (citations and footnotes omitted).

on a compact disk—or a player piano role—would be patentable as a "manufacture" simply because of the physical structure of the compact disk,[125] *a reductio ad absurdum* that goes a long way toward establishing that the majority's holding is actually untenable. Imagine for a moment that you are a judge of one of the numbered circuits hearing an appeal from a judgment of a district court upholding—or denying—a counter-claim seeking to have a patent like that in *Alappat* declared invalid.[126]

Which opinion in *Alappat* would you find more persuasive, that of the majority? Or that of Judge Archer?

B. State Street

In *State Street*, State Street Bank & Trust Co. initially obtained a judgment from the Federal District Court for Massachusetts declaring that a patent for a "Data Processing System[127] for Hub and Spoke Financial Services Configuration."[128] issued to Signature Bank was invalid because its subject matter was not patentable according to the provisions of 35 U.S.C. § 101. The Federal Circuit then reversed this decision of the District Court even though the patented invention was simply "a system[129] that allows an administrator to monitor and record

---

[125] Consider for example the discovery or creation of music, a new song. Music of course is not patentable subject matter; a composer cannot obtain exclusive patent rights for the original creation of a musical composition. But now suppose the new melody is recorded on a compact disc. In such case, the particular musical composition will define an arrangement of minute pits in the surface of the compact disc material, and therefore will define its specific structure. Alternatively suppose the music is recorded on the rolls of a player piano or a music box. Through the expedient of putting his music on known structure, can a composer now claim as his invention the structure of a compact disc or player piano roll containing the melody he discovered and obtain a patent therefor? The answer must be no. The composer admittedly has invented or discovered nothing but music. The discovery of music does not become patentable subject matter simply because there is an arbitrary claim to some structure.

And if a claim to a compact disc or piano roll containing a newly discovered song were regarded as a "manufacture" and within § 101 simply because of the specific physical structure of the compact disc, the "practical effect" would be the granting of a patent for a discovery in music. Where the music is new, the precise structure of the disc or roll would be novel under § 102. Because the patent law cannot examine music for "nonobviousness," the Patent and Trademark Office could not make a showing of obviousness under § 103. The result would well be the award of a patent for the discovery of music. The majority's simplistic approach of looking only to whether the claim reads on structure and ignoring the claimed invention or discovery for which a patent is sought will result in the awarding of patents for discoveries well beyond the scope of the patent law. *Id.* at 1553–54.

[126] The Federal Circuit does not have jurisdiction over an appeal from such a counter-claim. *See infra* note 153 and accompanying text.

[127] Recall that claims held to be unpatentable in *Benson* were also for a data processing system. *See supra* text accompanying note 34.

[128] *State Street*, 149 F.3d 1368.

[129] Which the Federal Circuit had the temerity to describe as a machine.

the financial information flow and make all calculations necessary for maintaining a partner fund financial services configuration."[130]

According to the Federal Circuit:

> "[A] partner fund financial services configuration essentially allows several mutual funds, or "Spokes," to pool their investment funds into a single portfolio, or "Hub," allowing for consolidation of, inter alia, the costs of administering the fund combined with the tax advantages of a partnership. . . . In particular, this system provides means for a daily allocation of assets for two or more Spokes that are invested in the same Hub. The system determines[131] the percentage share that each Spoke maintains in the Hub, while taking into consideration daily changes both in the value of the Hub's investment securities and in the concomitant amount of each Spoke's assets.[132]

And the "system" also makes other calculations, using—surprise! surprise!—a computer: "Given the complexity of the calculations, a computer or equivalent device is a virtual necessity to perform the task."[133] Thus the abstract idea[134] of partners investing in a partnership where an administrator frequently calculates each partner's interest using a computer is, according to the Federal Circuit, a patentable invention.

On rereading—time after time after time—this opinion of the Federal Circuit's, I still marvel that the judges of that court had the nerve to hold that the described "invention" was patentable in the face of the Supreme Court's statement in *Diehr* that "Excluded from . . . patent protection are . . . abstract ideas."[135] and the Supreme Court's holdings in *Benson* and *Flook* that using a computer to make calculations is not a patentable process even if the calculations have some element of novelty, an element that is completely lacking in the patent claims that were upheld in *State Street*. Using a computer,

---

[130] *State Street*, 149 F.3d at 1371.

[131] It is, of course, the system's administrator, not the system, who makes these determinations. The Federal Circuit seems to be suggesting here that mutual fund managers are simply parts of a machine, quite lacking in any of the attributes of humanity. If that were the case, then the teams of "girls" who acted as computers for the Manhattan Project were also mere machines. *See infra* Part VI.A.1.

[132] *State Street*, 149 F.3d at 1371.

[133] *Id.* at 1371.

[134] If it even rises to the level of an idea.

[135] *Diehr*, 450 U.S. at 185.

personal or otherwise, to process data is exactly what *Benson* holds is not patentable.[136]

What I find even more troubling is that the Federal Circuit made an extensive argument—and, in fact seems to have held *a la Alappat*—that Signature Bank's patent application claimed not a process, but rather a machine[137] and yet, in the end, admitted that it

---

[136] Of course, in *Benson* the Court was speaking of the "processing of data by program" (*supra* note 5), but that hardly strengthens the Federal Circuit's case in *State Street* since the computers used by the Hub's administrator would have to be programmed to do whatever it was that they were supposed to do. That there was no reference in the patent claims at issue in *State Street* to any program or formula raises the question of what the supposed invention was: was it just a means of doing business? *See infra* note 142.

[137] Here is the Federal Circuit's explanation of why the claimed invention was a machine:

> [The patent application] initially contained six "machine" claims, which incorporated means-plus-function clauses, and six method claims. According to Signature [the defendant patentee], during prosecution the examiner contemplated a § 101 rejection for failure to claim statutory subject matter. However, upon cancellation of the six method claims, the examiner issued a notice of allowance for the remaining present six claims on appeal. Only claim 1 is an independent claim. The district court began its analysis by construing the claims to be directed to a process, with each "means" clause merely representing a step in that process. However, "machine" claims having "means" clauses may only be reasonably viewed as process claims if there is no supporting structure in the written description that corresponds to the claimed "means" elements. *See In re Alappat* (citations omitted). This is not the case now before us. When independent claim 1 is properly construed . . . it is directed to a machine, as demonstrated below, where representative claim 1 is set forth, the subject matter in brackets stating the structure the written description discloses as corresponding to the respective "means" recited in the claims.
>
> 1. A data processing system for managing a financial services configuration of a portfolio established as a partnership, each partner being one of a plurality of funds, comprising:
>
> a) computer processor means [a personal computer including a CPU] for processing data;
>
> b) storage means [a data disk] for storing data on a storage medium;
>
> c) first means [an arithmetic logic circuit configured to prepare the data disk to magnetically store selected data] for initializing the storage medium;
>
> d) second means [an arithmetic logic circuit135 configured to retrieve information from a specific file, calculate incremental increases or decreases based on specific input, allocate the results on a percentage basis, and store the output in a separate file] for processing data regarding assets in the portfolio and each of the funds from a previous day and data regarding increases or decreases in each of the funds, [sic, funds'] assets and for allocating the percentage share that each fund holds in the portfolio;
>
> e) third means [an arithmetic logic circuit configured to retrieve information from a specific file, calculate incremental increases and decreases based on specific input, allocate the results on a percentage basis and store the output

makes no difference in determining whether the invention was patentable under § 101 whether the invention was claimed as a process or as a machine.[138]

After holding both that Signature Bank's "invention" was a machine the Federal Circuit went on to say that the "invention" was patentable even if it were not a machine:

> This does not end our analysis, however, because the [District C]ourt concluded that the claimed subject matter fell into one of two alternative judicially-created exceptions to statutory

---

> in a separate file] for processing data regarding daily incremental income, expenses, and net realized gain or loss for the portfolio and for allocating such data among each fund;
>
> f) fourth means [an arithmetic logic circuit configured to retrieve information from a specific file, calculate incremental increases and decreases based on specific input, allocate the results on a percentage basis and store the output in a separate file] for processing data regarding daily net unrealized gain or loss for the portfolio and for allocating such data among each fund; and
>
> g) fifth means [an arithmetic logic circuit configured to retrieve information from specific files, calculate that information on an aggregate basis and store the output in a separate file] for processing data regarding aggregate year-end income, expenses, and capital gain or loss for the portfolio and each of the funds. Each claim component, recited as a "means" plus its function, is to be read, of course, ... as inclusive of the "equivalents" of the structures disclosed in the written description portion of the specification.

Thus, claim 1, properly construed, claims a machine, namely, a data processing system for managing a financial services configuration of a portfolio established as a partnership, which machine is made up of, at the very least, the specific structures disclosed in the written description and corresponding to the means-plus-function elements (a)–(g) recited in the claim." *State Street*, 149 F.3d at 1371–72. Now that is some machine! Do you think that you could recognize it if you saw it? Is it the sort of thing that you could kick? Is it even the sort of thing that you could point to and ask, "What's that?"

[138] After holding that the "invention" was a machine the Federal Circuit on to say that:

> A "machine" is proper statutory subject matter under § 101. We note that, for the purposes of a § 101 analysis, it is of little relevance whether claim 1 is directed to a "machine" or a "process," as long as it falls within at least one of the four enumerated categories of patentable subject matter, "machine" and "process" being such categories. . . . *Id.* at 1372 (emphasis added). In other words, after entertaining us with the lengthy description of the most unmachinelike machine supposedly claimed by Signature Bank in its patent application, the Federal Circuit admits that, under its interpretation of the Patent Act, it does not make any difference whether the claim refers to a machine or a process. [And it is true, of course, that under the holding in Benson that it does not make any difference whether a process or a machine is claimed, for the claims in Benson were held by the Court to be unpatentable, even though at least one of them specified that the claimed invention was to run upon a computer containing a shift register and both could be used only in conjunction with a computer.]

subject matter. The court refers to the first exception as the "mathematical algorithm" exception and the second exception as the "business method" exception.[139] Section 101 reads:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefore, subject to the conditions and requirements of this title.

The plain and unambiguous meaning of § 101 is that any invention falling within one of the four stated categories of statutory subject matter may be patented, provided it meets the other requirements for patentability set forth in Title 35, *i.e.*, those found in §§ 102, 103, and 112, P2.

The repetitive use of the expansive term "any" in § 101 shows Congress's intent not to place any restrictions on the subject matter for which a patent may be obtained beyond those specifically recited in § 101. Indeed, the Supreme Court has acknowledged that Congress intended § 101 to extend to "anything under the sun that is made by man." *Diamond v. Chakrabarty*; see also *Diamond v. Diehr*, Thus, it is improper to read limitations into § 101 on the subject matter that may be patented where the legislative history indicates that Congress clearly did not intend such limitations. "We have also cautioned that courts 'should not read into the patent laws limitations and conditions which the legislature has not expressed.'"[140]

And then, at last, the Federal Circuit came to the issue of whether "mathematical algorithms" are patentable.

The Supreme Court has identified three categories of subject matter that are unpatentable, namely "laws of nature, natural phenomena, and abstract ideas."[141] In *Diehr*, the Court

---

[139] For more on the court's treatment of the "business method" exception, *see infra* note 142.

[140] *State Street*, 149 F.3d at 1372–73 (citations omitted). For an explanation of why *Chakrabarty* does not imply that computer programs are patentable, *see infra* Part II.D.

[141] *State Street*, 149 F.3d at 1373 (citing *Diehr*). The Federal Circuit then added: "Of particular relevance to this case, the Court has held that mathematical algorithms are not

> explained that certain types of mathematical subject matter, standing alone, represent nothing more than abstract ideas until reduced to some type of practical application, *i.e.*, "a useful, concrete and tangible result." [142]
>
> Unpatentable mathematical algorithms are identifiable by showing they are merely abstract ideas constituting disembodied concepts or truths that are not "useful."[143] From a practical standpoint, this means that to be patentable an algorithm must be applied in a "useful" way. In *Alappat*, we held that data, transformed by a machine through a series of mathematical calculations to produce a smooth waveform display on a rasterizer monitor, constituted a practical application of an abstract idea (a mathematical algorithm, formula, or calculation), because it produced "a useful, concrete and tangible result"–the smooth waveform. Similarly, in Arrhythmia Research Technology Inc. v. Corazonix Corp. we held that the transformation of electrocardiograph signals from a patient's heartbeat by a machine through a series of mathematical calculations constituted a practical application of an abstract idea (a mathematical algorithm, formula, or calculation), because it corresponded to a useful, concrete or tangible thing–the condition of a patient's heart. Today, we hold that the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price, constitutes a practical application of a mathematical algorithm, formula, or calculation, because it produces "a useful, concrete and tangible result"—a final share price momentarily fixed for recording and reporting purposes and even accepted and relied upon by regulatory authorities and in subsequent trades. . . .[144]

---

patentable subject matter to the extent that they are merely abstract ideas." *Id.* (citing *Diehr*, *Flook* and *Benson*). You may look as long as you like at those cases, but will never find the limitation that mathematical algorithms are unpatentable only to the extent that they are merely abstract ideas.

[142] *Id.* (citing *Alappat*). *Alappat* may say that, but is it a true statement of what the Court said in *Diehr*? And does it make any difference considering the holding in *Benson* that mathematical algorithms are not patentable? And, in any case, although the result, whatever it was, in *State Street* may have been useful, it certainly was neither concrete nor tangible.

[143] But, of course, the algorithm at issue in *Benson* was "useful."

[144] 149 F.3d at 1373 (citations and footnotes omitted).

The Federal Circuit also said toward the end of its opinion in *State Street* that:

> As an alternative ground for invalidating the . . . patent under § 101, the [trial] court relied on the judicially-created, so-called "business method" exception to statutory subject matter. We take this opportunity to lay this ill-conceived exception to rest. Since its inception, the "business method" exception has merely represented the application of some general, but no longer applicable legal principle . . . ., Since the 1952 Patent Act, business methods have been, and should have been, subject to the same legal requirements for patentability as applied to any other process or method.[145]

This ignores the fact that most business method patent claims, like those in *State Street*, are abstract ideas that involve only the processing of information without inducing any change in anything material. If computer programs, some of which are novel, unobvious, and useful are not patentable according to the Supreme Court, as I argue here, it is inconceivable that business methods like those disclosed in *State Street* should be patentable. It seems unlikely to me that anyone could read *Benson*, *Flook*, and *Diehr*, on the one hand, and *State Street*, on the other, and not conclude that the Federal Circuit believes that it has overruled the decisions of the Supreme Court. Need I remind you, once again, that the Federal Circuit lacks the power to do that?

C. The Supreme Court Begins to Wake Up

For many years by far the strongest argument for concluding that the Federal Circuit's decisions in cases like *Alappat* and *State Street* permitting software patents stated the law—and that the contrary decisions of the Supreme Court in *Benson*, *Flook*, and *Diehr* were, shall we say, obsolete—was that it seemed improbable that the Supreme Court would ever again consider an appeal from a decision in a case involving the validity or infringement of a patent, but would rather let those issues be decided by the supposedly expert judges of the Federal Circuit, which has almost exclusive jurisdiction over patent appeals from the lower courts.[146] After all, if Justice Holmes was right—as he pretty much was[147]—when he said that: "The

---

[145] *Id.* at 1375 (footnote omitted).

[146] *See supra* text accompanying note 111.

[147] At least from the point of view of a lawyer trying to advise a client as to what the law is.

prophecies of what the courts will do in fact, and nothing more pretentious, are what I mean by the law,"[148] then the decisions of the Federal Circuit as to the patentability of software did indeed state the law.

Although for years the Supreme Court has lived up to that prediction, it has shown signs of awakening from its slumber and shown a willingness to reverse the decisions of the Federal Circuit in at least one arcane area of patent law for in a case known as *Festo*[149] the Court showed a willingness to overrule the Federal Circuit in a case involving two arcane and contentious issues of patent law. More recently the Court has *sua sponte* in the *Metabolite* case[150] asked for a brief from the Solicitor General:

> expressing the views of the United States limited to the following question: Respondent's patent claims a method for detecting a form of vitamin B deficiency, which focuses upon a correlation in the human body between elevated levels of certain amino acids and deficient levels of vitamin B. The method consists of the following: First, measure the level of the relevant amino acids using any device, whether the device is, or is not, patented; second, notice whether the amino acid level is elevated and, if so, conclude that a vitamin B deficiency exists. Is the patent invalid because one cannot patent "laws of nature, natural phenomena, and abstract ideas"?[151]

Personally, I find it very difficult not to believe that that request indicates that the Court is ready to give the Federal Circuit a whack along the side of the head and reaffirm *Benson*, *Flook*, and *Diehr*.[152]

---

[148] O.W. Holmes, Jr., *The Path of the Law*, 10 HARV. L. REV. 457 (1897).

[149] Festo Corp. v. Shoketsu Kinzoku Kogyokabushiki Co., 535 U.S. 722 (2002). For an extensive and suggestive discussion of *Festo*, see John F. Duffy, *The Festo Decision and the Return of the Supreme Court to the Bar of Patents*, 6 SUP. CT. REV. 273 (2002).

[150] Laboratory Corp. of America Holdings v. Metabolite Laboratories, Inc., 543 U.S. 1185 (2005). The writ of certiorari was later dismissed as improvidently granted, almost certainly because the question that interested the Court had not been briefed below, but even so there was a strong dissent by three justices who would have held that the invention was not patentable because the application covered a "law of nature." 126 S. Ct. 2921.

[151] *Metabolite*, 543 U.S. at 1185 (citing *Diamond v. Diehr*). Note that the process of "noticing" whether the amino acid level is elevated could be conducted using a computer and that if the claim in question had specified using a computer, the patent at issue in *Metabolite* would have been as much a software patent as was the patent at issue in *State Street*.

[152] The fact that the claim in *Metabolite* does not refer to a computer or a computer program hardly weakens this conclusion. The process of "noticing whether the amino acid level is elevated" involves some sort of computation that can probably be done by head and hand without the aid of a computer, but that could also be done using a properly programmed computer.

The argument that the decisions of the Federal Circuit permitting software patents are the law, no matter what the Supreme Court has said about the matter, was strengthened[153] until recently by the fact that all appeals in patent cases were heard by the Federal Circuit so that there could never be a conflict between the circuits[154] in such cases.

But that is no longer true, for, since the Court's 2002 decision in *Holmes Group, Inc. v. Vornado Aircirculation Systems, Inc.*,[155] appeals from cases in which it is the defendant rather than the plaintiff who first raises the issue of the validity of a patent are no longer heard by the Federal Circuit.

Thus, even if the Supreme Court is not awakening of its own accord, someday there will be a case where the holder of a software patent sues for breach of contract because the defendant has ceased paying the royalty agreed upon in a patent licensing agreement and the defense will be that there was a failure of consideration because the patent was invalid. Now, according to Holmes Group, no matter what the trial court[156] decides about the validity of the patent, an appeal from that decision will not be heard by the Federal Circuit. In such a case, who would dare prophesize that the appellate court would follow the precedents of the Federal Circuit rather than those of the Supreme Court?

IV. THE ACADEMIC REACTION TO *BENSON* & CO.

Benson was decided before "computer law" or what is now, most unfortunately, called "cyberlaw"[157] had become a part of the curricula of law schools, so it is hardly surprising that *Benson* did not initially inspire a large number of law review articles. Since by the time Computing and the Law became a fashionable subject, *Benson* and

---

[153] And undoubtedly to a large extent justified.

[154] Or between the Federal Circuit and a state court of last resort.

[155] *See supra* note 111.

[156] The trial court will either be a state court or a Federal District Court that has diversity jurisdiction or has subject matter jurisdiction where the subject matter is not the patent law.

[157] It is fashionable I fear for courses about computers and the law to dwell not on the fundamental legal issues involving the process of computing, but rather on the more fashionable, but less fundamental, issues involving the Internet. One consequence of this is the proliferation in law schools of courses with names like "Cyberlaw" or "Law and Cyberspace," "Cyberspace" being one of those mind destroying metaphors that I wish would just go away, The term "cyber"—Greek for governor or steersman—that is part of the term "cybernetics," refers to the study of control mechanisms. Even for courses that are about the Internet "Cyberlaw" is not an appropriate name, for the Internet is not a space in which a steersman—or even a computer program—can navigate or over which a governor could exercise control. Whatever control is exercised over the Internet, is exercised by people and devices and their programs that are at the edges of the network, not within the "space" of the network itself.

company were ancient history, it is hardly surprising that little attention has been paid to them recently in the groves of legal academia. There the assumption appears to be that software is patentable as a matter of law and that the only issue relating to software patents that merits discussing is whether or not such patents can be condemned or justified in economic terms.

There are, however, two articles from the mid-eighties and one from 1990 about *Benson*—and, to a much lesser extent, its progeny—that merit discussing here.

A. The Patentability of Algorithms

Donald S. Chisum, the author of the treatise Chisum on Patents,[158] wrote an extremely critical analysis of the Supreme Court's decision in *Benson* entitled The Patentability of Algorithms[159] that was published in a symposium on The Future of Software Protection, published by the University of Pittsburgh Law Review in 1986.

Chisum's major criticism of the Supreme Court's decision in *Benson* appears to be that the Supreme Court in that decision failed to follow the established principles of patent law as Chisum—and probably most patent lawyers at the time—understood them. Here is how Chisum summarized his argument:

> New and useful algorithms, including mathematical algorithms, should constitute subject matter eligible for patent protection. Yet, the current state of the law is that "mathematical" algorithms "as such" or "in the abstract" do not constitute patentable subject matter—at least not in theory.

In this Article, I hope to demonstrate the weakness of the second, descriptive statement and the soundness of the first, normative statement. My case consists of the following (nonalgorithmic) line of analysis. First, prior to the Supreme Court's decision in *Gottschalk v. Benson* in 1972, the lower courts had, for appropriate reasons, rejected earlier doctrine on the nonpatentability of so-called mental steps and established a firm basis for the patenting of algorithmic ideas. Second, the *Benson* decision, which held that mathematical algorithms could not be patented, was poorly reasoned and stemmed from an antipatent judicial bias[160] that cannot be reconciled with the

---

[158] DONALD S. CHISUM, UNDERSTANDING INTELLECTUAL PROPERTY LAW (New York, NY: M. Bender, 1996).

[159] Chisum, *supra* note 32.

[160] As opposed to a pro-patent, patent lawyer bias? Whose bias, if bias it be, is more likely

basic elements of the patent system established by Congress. Third, the awkward distinctions and seemingly irreconcilable results of the case law since *Benson*, including the Supreme Court's decisions in *Parker v. Flook* and *Diamond v. Diehr*, are a product of the analytical and normative weakness of *Benson* itself. Finally, an examination of the policy implications of extending patent protection to new algorithms will show that such extension will not harm the creation and dissemination of knowledge in computer science and other areas of technology and will in fact provide much needed additional incentives for investment in computer software development.[161]

As you probably can guess, I do not believe that Chisum came anywhere near to proving his case. But he certainly supplies a much more detailed history of the relevant patent law cases than I have done here. If you are more interested in patent law than I am, you really should read his entire article.

Here is part of what Chisum has to say about the relevant pre-*Benson* cases and Patent Office practice:

> Early case law and Patent Office practice developed three vaguely defined limitations on the scope of patentable subject matter that would later cast doubt on the patentability of algorithms designed for use in computer programming. The limitations relate to (1) business systems, (2) printed matter, and (3) mental steps. All three limitations suffer from a common infirmity—the absence of a firm footing in either statutory language or well-reasoned, extrastatutory policy.
>
> A. Business Systems
>
> It is thought to be black-letter law that a "system of transacting business disconnected from the means for carrying out the system" does not constitute patentable subject matter. The 1908 decision by the Second Circuit so stating offers no precise reason for such an exclusion. . .[162]
>
> B. Printed Matter
>
> Early court decisions held that an invention consisting of the arrangement of information on a substrate, however new and

---

to prevail in determining how patent cases should be decided?

[161] Chisum, *supra* note 32, at 960–62.

[162] The Federal Circuit completely rejected the business systems exception in its opinion in *State Street*. *See supra* Part III.B.

useful, could not constitute patentable subject matter unless the invention called for a new relationship between the information and the substrate. This limitation was closely related to that on business systems since most attempts to patent "printed matter" involved arrangements of information designed to implement some business system. But the decisions give no reason for excluding printed matter independent of a desire not to allow a subversion of the exclusion of patents on business systems. Recent decisions question the scope of the printed matter limitation without directly repudiating it. . . .

C. Mental Steps

The mental steps doctrine has a more complicated history. Early decisions struggled with the definition of a "process" or "art." It was established that, on the one hand, one could not patent a newly-discovered "principle" or "law of nature" in the abstract, but that, on the other hand, one could patent the application of such a principle to create a new product or method. In developing the distinction between unpatentable principles and patentable applications, some decisions suggested (without actually holding) that "process" must involve the transformation of matter in some way. This definition, if rigorously applied, might limit processes to chemical and some mechanical processes. And indeed the earliest "mental steps" decision in the Patent Office took this position. In *Ex parte Meinhardt*, the Commissioner ruled that a "system for spacing free-hand letters on a page" did not constitute patentable subject matter, relying solely on case law concerning the definition of processes.

. . . .

With the coming of the computer age, processes involving "mental steps" no longer would necessarily be performed by the human brain but rather could be performed by the marvelous new computing machines. Relying in part on the mental steps doctrine, the Patent Office as a matter of policy would not allow patents on "software" or computer programming inventions. But, in a brief three-year period beginning in 1969, the Court of Customs and Patent Appeals (C.C.P.A.), in reviewing Office actions rejecting software

> patent applications, dismantled the mental steps doctrine. . . .[163]

Chisum then included as an entire section in his article a discussion about the "Definition of an Algorithm" where he claimed that:

> In *Benson*, the Supreme Court recited a definition of an algorithm:
>
> > A procedure for solving a given type of mathematical problem is known as an "algorithm."[164] The procedures set forth in the present claims are of that kind; that is to say, they are a generalized formulation for programs to solve mathematical problems of converting one form of numerical representation to another. From the generic formulation, programs may be developed as specific applications.
> >
> > The Court erred both in implying that algorithms relate only to mathematical problems[165] and in characterizing the method involved in *Benson* as directed to "mathematical" problems. It is true that algorithms are often devised to solve problems of a mathematical nature.[166] But algorithms may also be

---

[163] Chisum, *supra* note 32, at 964–71 (citations and footnotes omitted).

[164] *Nota bene*, the Court did not say that an "algorithm" is a procedure for solving (only) a given type of mathematical problem.

[165] Where did the Court imply that? There is some confusion here, for it is clear that the Supreme Court in *Benson* was dealing only with algorithms for processing data.

[166] At this point, Chisum inserts the following footnote:

> Perhaps the most famous is the algorithm derived from Euclid for finding the greatest common divisor of two positive integers a and b. One text relates the algorithm as follows:
>
> 1. Compare a and b (a = b, or a < b, or a > b). Go on to 2.
> 2. If a = b then either is the greatest common divisor. Stop the computation. If a not = b go on to 3.
> 3. Subtract the smaller from the larger number and write down the subtrahend and the remainder. Go to the next instruction.
> 4. Assign symbol a to the subtrahend, and symbol b to the remainder. Return to direction 1.
> 5. The procedure is repeated until a = b. Then the computation is stopped.

> devised to solve all sorts of nonmathematical problems.[167] The method at issue in *Benson* meets all of the . . . definitions of an algorithm . . . . But is the method one for solving a "mathematical problem" as the Court states?[168] It would seem that a conversion of decimal numbers to binary-coded numbers to binary numbers is a "mathematical" problem only in a very loose sense. It is more properly described as a translation problem—comparable to converting temperature values from Fahrenheit to Celsius.[169] The algorithm involves some arithmetic steps (such as adding in binary form), but the problem solved is not a mathematical one (such as finding the greatest common divisor of two numbers or a trigonometric function).[170] The imprecision of the Court in characterizing the algorithm before it in *Benson* created uncertainty as to the scope of the exclusionary rule that it upholds.[171] After all, what *Benson* dealt with is an algorithm that was used to process information—to convert one representation of data to another representation.

Chisum then purports to "unravel" the "confusion" in the *Benson* opinion,[172] claiming that "[T]he reasoning in *Benson* monstrously bad."[173]

This "unraveling" consists primarily of describing the cases—all from the Supreme Court—cited in *Benson* and then claiming that they do not support Justice Douglas's judgment in that case.[174] Chisum's

---

Chisum, *supra* note 32, at 976 n. 56.

[167] This certainly is true: for example, there might be an algorithm solving the problem of how to cook a seagull or how to cure rubber.

[168] As long as we know the exact nature of the algorithm, why should we worry about whether it is "mathematical" or not?

[169] And that, of course, is a classic example of a mathematical problem! I remember studying it early on if high-school algebra.

[170] Translating binary coded digits to pure binary digits (or "bits") most certainly is a mathematical problem. Mathematics, after all, covers much more than arithmetic; in particular it includes the Boolean logic that is the basis for all computer programs. *See infra* Part VI.A.5. But even were this not the case, one should note that shifting a binary representation of number to the right (or left) is the equivalent of multiplying (or dividing) that number by two. *See infra* note 391 and accompanying text.

[171] Chisum, *supra* note 32, at 977 (footnotes omitted). Would it not be more accurate to say that the alleged imprecision allowed the patent bar to raise doubts about the scope of that rule?

[172] Which I submit is more in Chisum's head than in the opinion itself.

[173] Chisum, *supra* note 32, at 977–78.

[174] That the cited cases do not fully support Justice Douglas's conclusion is, of course,

arguments are perhaps a legitimate criticism[175] of the reasons Justice Douglas gave for his conclusions but they do not in any way address the holding in that case that algorithms for the processing of information are not patentable. And, of course, it is the holding in *Benson* that is binding on the lower courts and that will be followed by the Supreme Court unless it decides to overrule it. However much Chisum and his allies may feel that *Benson* was wrongly decided, it is still protected by the doctrine of stare decisis.

Here are what I take to be the most persuasive part of Chisum's critique:

> After quoting . . . three cases, *Benson* offers a dogmatic statement: "Phenomena of nature, though just discovered, mental processes, and abstract intellectual concepts are not patentable, as they are the basic tools of scientific and technological work." This statement makes three assertions—two explicit and one implicit. The explicit assertions are that (1) natural phenomena, mental processes, and abstract intellectual concepts are to be lumped together as unpatentable subject matter, and (2) the reason for such nonpatentability is that all are "basic tools" of technological work. The implicit assertion is that an algorithm . . . is a phenomenon of nature, mental process or abstract intellectual concept. None of the three assertions will bear up fully under analysis.
>
> First, a mathematical or other algorithm is neither a phenomenon of nature nor an abstract concept.[176] The *Benson-Tabbot* algorithm is very much a construction of the human mind.[177] One cannot perceive an algorithm in nature.[178] The algorithm does not describe natural phenomena

---

hardly surprising; after all *Benson* dealt with issues that had never been considered before by the Supreme Court.

[175] Though I do not think so.

[176] As to whether there is a distinction between a phenomenon of nature and one of mathematics, consider the philosophical question of whether the law that "one and one makes two" is a phenomenon of nature or one of mathematics. Should the answer have any bearing on the question of whether mathematical algorithms are patentable? And what on earth is a mathematical algorithm if it is not an abstract concept?

[177] Which human mind?

[178] There are those who know far more about nature than any law professor who seem to have difficulty in perceiving anything other than algorithms in nature. *See* SETH LLOYD, PROGRAMMING THE UNIVERSE (2006); *see also* DANIEL C. DENNETT, DARWIN'S DANGEROUS GAME (1995).

> (or natural relationships).[179] Indeed, it does not describe anything other than a series of operations to be performed by a machine or possibly by a human being.[180] . . . Neither is the algorithm fairly described as an "abstract intellectual" concept (such as "all persons are created equal" or "for every action there is an equal and opposite reaction").[181] Algorithms are by definition highly specific rather than abstract.[182]
>
> Second, there is no basis for lumping together phenomena of nature and abstract concepts with "mental steps." A process consisting partially or wholly of "mental steps" does not exist in nature[183] and can be quite specific.[184] The Court's reference to "mental processes" is disturbingly terse. Arguably, the *Benson-Tabbot* algorithm as stated in claim 13 (though not in the machine claim 8) did involve "mental processes" if the claim is construed as covering human and not just machine implementation. . . .[185]
>
> Third, the unpatentability of "phenomena of nature" is not so clear and self-evident as the Court in *Benson* would have us believe. . . .[186]
>
> If the trio of "unpatentables" are in fact property excluded from the patent system, it is for distinct reasons and not their "tool" status. If natural phenomena are excludable, it may simply be because they are not "new" as the patent statutes require. Abstract concepts are excludable because of the disclosure and clear claiming requirements of the patent statutes.[187]

---

[179] If an algorithm describes something, which I rather doubt, what is it that it describes? Something super natural?

[180] Aren't those operations natural?

[181] Is Chisum claiming here that the algorithms describing how to do long division or the one describing how to find the square root of an arbitrary number are not abstract intellectual concepts?

[182] Is not the number "2" very specific and, at the same time, completely abstract?

[183] So when you or I think of something, we are doing something unnatural?

[184] What earthly difference dose it make whether mental steps are specific, as I suspect they usually are?

[185] What difference does it make whether a human being uses pencil and paper, an abacus, or a computer in implementing an algorithm in order to process data?

[186] Now, if that be true, it hardly vitiates the Court's holding in *Benson*, now does it?

[187] Chisum, *supra* note 32, at 980–84. Chisum then claims that the decisions in *Flook* and *Diehr* are confusing because of the defective reasoning in *Benson*, gives a short history of some subsequent cases decided by the Court of Customs and Patent Appeals, and finally argues on policy grounds that software patents are a good thing. In my opinion none of this need concern

Again, while I believe that Chisum's critique of Justice Douglas's presentation of the Court's reasons for its holding in *Benson* may have some merit—the opinion is, after all, quite difficult to parse—I cannot find any argument in his article that suggests that the holding in that case is wrong.

B. The Models Are Broken, the Models Are Broken!

Allen Newell, one of the earliest computer scientists and one of the first to explore the field of artificial intelligence and what today is known as cognitive science, wrote, in the same symposium, a response to Chisum's article entitled The Model's Are Broken, The Models are Broken![188] In this article Newell explains—or at least begins to explain—why we lawyers find the issues in *Benson* so confusing and also, I think, why Justice Douglas's opinion in that case is not as clear as one—especially one who thinks it correct—might wish.

Still, as we shall see, Newell failed to discuss what I find to be the most confusing aspect of *Benson*, the fact that, though the case involved an algorithm for processing information, few of us can say with any certainty what information is,[189] a matter that Newell did not discuss.[190]

Newell claimed to be agnostic on the question of whether software patents should be issued.[191] His position was that, as the title of his article suggests, our models—our ideas—about the nature of algorithms and related matters are going inevitably to plunge us into confusion, no matter whether we allow software patents or forbid them.

Thus Newell says at the beginning:

> Professor Chisum's Article launches a full scale attack on *Gottschalk v. Benson*, which held—erroneously in Chisum's view—that algorithms are not patentable. Chisum may well be right that the *Benson* case has brought on much analytic confusion in the software patent area. He may also be right in

---

us here.

[188] Allen Newell, *The Model's Are Broken, The Models are Broken!*, 47 U. PITT. L. REV. 1023 (1986).

[189] *See infra* Part VI.C.

[190] My suspicion is that as a computer scientist Newell was so immersed in the understanding that computers are machines which process data, *i.e.*, information, that he did not appreciate how confusing that can be to those who think of machines as gadgets that produce material objects like automobile engine blocks or properly cured rubber.

[191] It seems to me though that most of his remarks suggest that such patents not be issued.

> supposing that, if *Benson* had only been decided differently, the specific confusion that occurred in its aftermath would not have materialized. It seems to be his view that a different holding in *Benson* would have brought about analytic sweetness and light.
>
> My point is precisely to the contrary. Regardless how the *Benson* case was decided—whether that algorithm or any other was held patentable or not patentable—confusion would have ensued. The confusions that bedevil algorithms and patentability arise from the basic conceptual models that we use to think about algorithms and their use. . . .[192]

Now that statement should perhaps be taken as a warning that Newell's comments are primarily directed at algorithms in general, not those, like that in *Benson*, that can be used only in data processing. Nor does Newell discuss, at least not directly, the issues in *Flook*,[193] a case that involved a "mathematical formula" rather than an algorithm.[194] Newell starts his discussion with a section entitled "The Nature of Algorithms" in which he first points out that "the one thing that does not present conceptual difficulties is the notion of an algorithm itself."[195]

According to Newell:

> The first confusion is using involvement with numbers as the hallmark for distinguishing mathematics from nonmathematics, as an aid to determining what is an algorithm. This confusion is easily cleared up. Algorithms are

---

[192] *Id.* at 1023 (footnote omitted).

[193] That is hardly surprising since Chisum did not discuss *Flook* at any length.

[194] *But see supra* note 87 and accompanying text where the Court in *Flook* equates a mathematical algorithm with a mathematical formula. *See also infra* Part VI.B.1 where it is contended that all computer programs, including those that solve numerical problems are basically algorithms and that all computers programs can be viewed as performing numerical calculations.

[195] Newell, *supra* note 188, at 1024. He may be being too sanguine, but Newell then continues:

> An algorithm is an unambiguous specification of a conditional sequence of steps or operations for solving a class of problems. This definition is perfectly reasonable, it is not arcane, and I believe we can all live with it. The confusion, then, is not in the nature of algorithms. It is all the things around it that get confused.

Id.

> certainly mathematical objects. That is an acceptable model. However, mathematics deals with both nonnumerical things and numerical things. Correspondingly, there are both numerical and nonnumerical algorithms. Therefore, any attempt to find a helpful or cutting distinction between mathematics and nonmathematics, as between numerical or nonnumerical, is doomed.[196]

This implies, of course, that if mathematical algorithms are held not to be patentable, as they were in *Benson*, then non-mathematical algorithms should be treated in the same fashion. Note, however, that that does not mean that algorithms that process data and those that process more material things, like rubber, for example, need be treated in the same way. One can also—contra Newell—make a strong argument that, since digital computers deal only with strings of "bits" that can always be interpreted as a binary representation of a number, all algorithms implemented with a digital computer must be numerical algorithms. Those two points seem at least somewhat to abate the confusion that Newell finds inherent in Chisum's interpretation of the issues in *Benson*.

Newell goes on to say:

> Next consider algorithms and mental steps.[197] The main line of progress in psychology for the last thirty years (called cognitive psychology) has been to describe human behavior as computational. We model what is going on inside the thinking human brain, as the carrying out of computational steps. Therefore, humans think by means of algorithms. Sequences of mental steps and algorithms are the same thing. Any attempt in the law to make distinctions that depend upon contrasting mental steps versus algorithms is doomed to

---

[196] *Id.* Newell then explains:

> Indeed, in the mid-1930s the central argument of a famous proof, by the mathematician Kurt Godel, involved showing that all of logic is a part of number theory. The scheme, which ever after has been called Godel numbering, assigns an integer to each logical expression, such that all the truths in logic become theorems about the integers. Although lawyers need never become acquainted with Godel numbering, they should realize there is an underlying identity between the numerical and the nonnumerical realms that will confound any attempt to create a useful distinction between them.

Id. at 1024–25.

[197] Recall that, though Chisum discussed the mental steps doctrine at some length, that doctrine was never mentioned, at least by name, in the Supreme Court's opinion in *Benson*.

> eventual confusion. . . . Any attempt to erect a patent system for algorithms that tries to distinguish algorithms as one sort of thing and mental steps as another, will ultimately end up in a quagmire. Just avoiding the use of this distinction is only half the story. An identity between algorithms and mental steps leads to such questions as whether you can keep people from thinking patented thoughts. . . .[198]
>
> Is not this a powerful argument against the patentability of algorithms: if one cannot patent mental steps—or ideas—then one cannot patent algorithms—at least not those algorithms that can be implemented as a series of mental steps?[199]

In the next section of his article, entitled "Algorithms and Basic Truths," Newell says:

> [T]he natural question is whether algorithms are to be considered either natural laws or mathematical truths,[200] hence not to be encouraged by patents; or whether they are inventions that jump the gap from such laws and truths to application, hence to be encouraged by patents. One view comes from observing how practical algorithms can be. They are directly related to use in specific tasks, tasks that can be of the utmost value. That would seem to place them as devices to jump the gap to application, hence as patentable.
>
> But a more confounding answer flows from the general nature of computer science. All of computer science is directly related to use. There is essentially no gap, no matter how pure or basic the science is. . . .
>
> Let us pursue the consequences of patenting algorithms. Would it have been possible to patent the integers? "No," comes the reply—integers are mathematical truths. Even I know that. . . .[201] So the integers cannot be patented. But

---

[198] Newell, *supra* note 188, at 1025.

[199] *See also infra* text following note 203.

[200] Note that Newell here treats natural laws and mathematical truths as being equivalent, overlooking the potential confusion that can arise—at least in some people's heads—when natural laws, which deal with material objects like galaxies and atoms and seagulls, are equated with mathematics, which is not concerned with such matters.

[201] Newell then qualifies this answer:

> Actually, the integers are abstract, mathematical objects, and mathematical truths are things like," Between every two consecutive odd integers there is exactly one even integer." But let that pass.

> certainly one might contemplate patenting addition. "Well, that is still not true," comes the reply—addition is defined to be a mapping of pairs of integers (the addends), to integers (the sum). . . .[202] Doing addition is accomplished by carrying out an algorithm. If algorithms are patentable, then I can keep you from doing addition with the algorithms invented for it. There would be ever so many things that the poor would not be able to do, such as add up their grocery bill.[203]
>
> . . . .[204]

Newell concludes this section by remarking that:

> There is one last way to avoid the swamp that seems to await the patenting of addition methods. We can focus on the primeval character of addition. Yes, it might have been a problem, but not now. Addition cannot be patented, because it is already in the public domain. Moreover, it is special—we will not see its like again in terms of generality and pervasiveness. But this answer will not do. Computer science is nothing but a breeding ground for new algorithms, and computer science is hardly out of its swaddling clothes. Algorithms of immense generality and scope will continue to emerge for as long as the science endures. Examples are easy to come by. The simplex algorithm for doing linear programming was invented by G. Dantzig in 1948. It was, until recently, the only practical algorithm for solving a huge class of management and production problems. The fast Fourier transform was invented by J. W. Cooley and J. Tukey in 1965. It essentially created the entire field of digital signal processing. Although considering the patenting of algorithms

---

Newell, *supra* note 188, at 1027.

[202] Once again Newell adds a qualification:

> That is not even a truth, that is just a definition. Ah, but if you want actually to do addition—that requires doing a sequence of things, not to the integers, which are abstract (so you cannot do things to them anyhow), but to some representation of the integers.

Id.

[203] *See also infra* text accompanying notes 204–06.

[204] Newell, *supra* note 188, at 1027.

> for addition may seem a bit melodramatic, it is perhaps not entirely unrepresentative.[205]

That is an extremely important point. The simplex algorithm, for example, is certainly novel enough and useful enough to be patented if such algorithms can indeed be patented, but if it were patentable then[206] Newton's invention of the calculus would have been patentable.[207] And if Newton had a patent on the calculus, no subsequent mathematician or natural scientist could[208] have done any serious work without Newton's permission. Nor could any student have studied mathematics beyond the high-school level legally without Newton's consent—imagine the police coming to your door and informing you that your daughter has been arrested for practicing calculus without a license.

The next section is entitled "The Embodiments of Algorithms." It is the most technical portion of Newell's article, but it is, I submit, well worth our consideration.

> Let us turn to the embodiments of algorithms. Computer science understands well the essential feature of a digital computer. First, there is a machine (call it the operations-machine) that can perform any of a collection of operations. For standard computers these are mostly operations that modify, store or retrieve data; but many other sorts of operations are possible. This is a real machine, by anyone's standards. However, when turned on, it does not actually do any of its operations, but awaits some signals from the outside to evoke them. Second, there is a specification (the program) of what behavior is desired, that is, what operations are to be performed and in what sequence. This specification is essentially a textual document,[209] although it is not encoded as marks on paper, but as marks in some other medium. Lastly, there is an interpreter, which is also a real machine by anyone's standards. Except that electrons are very quiet it would clank. If you feed the specification into the interpreter, the interpreter will send signals to the operations-machine to make it carry out the sequence of steps specified. By now

---

[205] *Id.* at 1028.

[206] Provided that the same rules of patent law existed in Newton's day as exist today.

[207] Unless Leibnitz were able to establish priority.

[208] As long as the patent was in force.

[209] In other words, it consists of information or, to be more precise, data, just as any other text does. *See infra* Part VI.C.

> there is nothing obscure about this arrangement and I daresay you are all familiar with it.[210]
>
> There may be nothing obscure about the arrangement, but several remarkable consequences flow from it. The one of interest to us can be called the transfer of creativity. The operations-machine has a certain scope for action. It is a physical arena in which one can imagine inventing how the machine might do new things and then reducing this to practice. What the interpreter does is to transfer all these possibilities for action to the specification. Going through the interpreter does not lose any of the capabilities of the operations-machine or behavior. Consequently, nothing is added to the nature of creation and invention beyond that which happens in the programming language. . . .[211]

In more modern—although sometimes misleading—terminology the interpreter takes the "source code" written by a programmer and converts it to "machine code." that is implemented by the computer.

Newell then says:

> What is true of programs is true of algorithms. An algorithm is just an abstract program, which is to say, just an abstract specification. The abstraction involved in an algorithm concerns relatively routine matters—what exact representation of the data to use, how to modify these representations in detail to accomplish the primitive steps specified by the algorithm, etc. These are important and their choice may make important efficiency differences. But the essential issues of what is to be done are reserved to the algorithm. Indeed, the whole point of writing an algorithm is to convey the essential of what the operations-machine is to do. Thus, any attempt, for the purposes of locating creativity and invention, to distinguish between the algorithm and any particular embodiment of it turns out to be extremely difficult.

---

[210] To modern readers this seem more obscure than Newell thinks, but that is only because today we are accustomed—at least if we use personal computers—to using the same chunk of hardware as both the operations-machine and the interpreter. *See infra* note 400 and accompanying text.

[211] Newell, *supra* note 188, at 1029.

> This also explains why algorithms cannot be distinguished from programs. . . [212]

This is an extremely important point, for if algorithms cannot be distinguished from programs, then the rule established in *Benson* that algorithms that can be implemented on a digital computer are not patentable must also apply to computer programs in general. If one cannot tell one from the other, then the fact that one cannot be patented implies that, for any practical purpose, the other cannot be patented.

And, then, in a section entitled "The Form of Algorithms" Newell says:

> Let us talk about how to recognize an algorithm when you see one. The definition of algorithm at the beginning says it determines a sequence of steps. Thus, we might expect an algorithm to be in the form of a procedure—do this, and then do this, and then do this, and so on. Anyone can tell a procedure, for it lays out exactly the steps to be taken. . . . So

---

[212] *Id.* at 1029. Newell adds:

> Computer science makes the distinction all right, but to express degrees of abstraction. An algorithm is more abstract than a program. Given an algorithm, it is possible to code it up in any programming language. You might think that a program should be something like an algorithm plus implementation details. Thus, you examine the text of a purported algorithm—if you find an implementation detail, you know it is a mere program. But it is not so, at least not in any way useful to the law. For a program is also abstract. It is a general specification for action that is interpreted for a particular occasion by the interpreter. The program is missing certain details, which must be added when the program is to be run (by other programs variously called compilers, assemblers and loaders). The root difficulty is that one man's detail is another man's essential. It all depends on the purposes. This analysis can be turned around. Since an algorithm determines the sequence of steps to be performed, then there must exist some, perhaps powerful, interpreter than can go directly from the specification of the algorithm to carrying out the steps. The existence of such an interpreter is the acid test of whether an algorithm has really been given. But then the algorithm in this high-level representation looks just like a programming language of a high-level and powerful interpreter. There are certainly matters of degree—algorithms are conventionally more abstract than programs. But there are no separations of kind, even though in the current art we often do not know enough to construct the powerful interpreters. This state of affairs, by the way, is reflected currently in the automatic programming field, where they refuse to draw any hard and fast line between systems to design algorithms and systems to design programs. From their point of view, it is just a continuum of design tasks. The computer science field maintains a distinction (so you can buy textbooks on algorithms and textbooks on programming), but it is more the distinction between what specifications to write to do tasks of interest versus the details of particular schemes for writing the specifications.
>
> *Id.* at 1029–30.

that is a model for how to tell an algorithm when you see one. Look at the text and examine its form to see if it is a procedure. Historically, that describes pretty closely what programming languages (Fortran, Ada, Cobol, assembly languages, etc.) and informal notations for algorithms (in textbooks, etc.) are like.

But alas, for our models, the reality of computer science moves on. This reality leads to conceptually richer ground that is highly productive for both theory and application. But it destroys the clean model whereby an algorithm could be recognized by its having a procedural form. Computer science takes an algorithm to be any specification that determines the behavior of a system. These specifications can be of any kind whatsoever as long as they actually provide the determination through the interpreter. Consequently, the form of the specification need no longer be procedural. Sequences of steps must march out after interpretation, but sequences of steps need not march into the interpreter.

This is hardly an idle possibility. We now have languages for writing algorithms that look very different from a sequence of steps. For instance, in some programming systems one simply provides a set of constraints that are to be satisfied by the ultimate actions, and the interpreter (or compiler) determines what actions are needed to satisfy them, and then executes them. A set of constraints does not look like a step-by-step procedure, but it is just as good as one, because it determines the steps. Other cases are no longer even esoteric and have moved well beyond the realm of research. The form of many expert systems is simply a collection of if-then rules that provide the knowledge that is needed to perform a task. There is no easy way of seeing such an expert system as a sequence of steps—except that the rule interpreter determines at each moment one rule to fire. Some of you may be familiar with the rhetoric of PROLOG, a prominent expert-system language, "You don't have to say how it's done, you can simply give to the system the knowledge about the task and it will go and do it." The rhetoric is a little overblown, but it illustrates how difficult it will be to detect whether some text is an algorithm from its form alone. If all that counts is the

> knowledge, then the specifications can look declarative or procedural or any other way. . . .[213]

Fortunately we do not have to recognize an algorithms when we see one. Since algorithms and computer programs are interchangeable, as long as we can recognize a computer program—and the proof that something is a computer program is that it can be run on a computer—we can be sure that it, and any of its specifications, are not patentable under the rule established in *Benson*.

And then, in the final section of his article, entitled "Increasing the Invention of Algorithms" Newell wonders into that slough of despond where economic arguments grow.

> My final example concerns whether patenting algorithms will lead to more or less innovation in the software field. The standard economic model lying behind the patent system, which was in evidence in several places in Professor Chisum's Article, is that inventors produce inventions that, in turn, produce more of the consumables[214] that society desires. Giving control of the invention to the inventors increases the price of the ultimate products to society, over what would be the case if the inventions were available for all to use. But this cost is more than offset by the increased number of inventions that become available to society, due to the incentive to inventors. These inventions raise the net productivity of the economy.[215]
>
> Consider an alternative model, in which inventions produce, not consumables, but "inventables." That is, suppose the primary effect of every product is to enable additional inventions. There will be consumption as well, but mostly new invention based on ownership of the products. To be clear about the key assumption: New invention comes from the products produced by the original invention, not from the original invention itself. The price increase due to the patent monopoly will restrict the amount of product sold, just as in the original model. But now this implies that the number of

---

[213] *Id.* at 1031–32.

[214] "Consumables" is certainly not the right word here. Computer programs are not consumable, for they can never be consumed. One can make and run as many copies of a program that one wants, and can give them all away, and the program as such still exists in all its unconsumed ontic glory.

[215] How on earth could anyone ever tell whether the cost is offset by the value of the increased number of inventions, or even that there would be an increased number of inventions.

> new inventions that occur will also be restricted. As long as there is some baseline flow of inventions in society, then it is possible that the loss of inventions to society from the monopoly price restrictions will more than offset the gains in inventions from the financial encouragement. The patent system could discourage invention rather than encourage it.[216]
>
> This may seem like an extreme alternative model. However, some computer communities may approximate it. There, people take each other's programs freely, then enhance them, and then pass them on to others, who do more of the same. Of course, they also use them as well. But consuming the behavior of a program does not consume the program.[217] Furthermore, it is the possession of the previously invented program that permits the new invention to occur. For the new inventor adds, modifies, enhances and reshapes the existing system, mostly in small ways, though occasionally substantially. The capabilities of the system evolve and grow. The motivations for such enhancements are partly that one benefits from the inventions themselves, for one gets to use the enhanced system. But the motivations are also partly those that keep the artist and the mathematician creating—it becomes a medium of expression and a coin of the realm. If patentability implies that mostly what is used is left untouched and unenhanced, then the total improvements in the community's software may well decrease, even though some people are induced to work harder at innovating to capture the rewards from patents. They must do their inventing from a poorer base.[218]

And then Newell adds:

> I do not seriously defend this alternative model. My goal is more modest – to point out that the world of algorithms and computers may have a different character than the standard economic model of incentives that underlies the patent law. Even this model may be broken.[219]

---

[216] Except for the last sentence, I simply cannot understand what Newell is trying to say in this paragraph.

[217] Exactly!

[218] Newell, *supra* note 188, at 1033–34.

[219] *Id.* at 1034.

If Newell was not prepared to defend his alternative model, the fact is that it seems a prescient description of the world that is occupied by those who produce what is known as "free" or "open" software, software that includes the Linux and BSD operating systems, the GNU compilers, editors and other utilities, the World Wide Web and the Apache servers that support it, and even the protocols that underlie the Internet itself.

C. Benson *Revisited*

In 1990 Pamela Samuelson, one of the leading experts—and probably the leading expert—on computer law, published, before the Federal Circuit made its decisions in *Alappat* and *State Street*, an article entitled *Benson* Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions.[220] I suspect that if this article had been given the attention that it deserved back then I would not now be writing this article, except perhaps to clarify a few details here and there.[221]

Samuelson's article begins:

> For most of the past twenty-five years, it was widely believed that computer program-related inventions were rarely, if ever, patentable. The 1972 Supreme Court decision, *Gottschalk v. Benson*, in which the Court ruled that a computer program algorithm is not patentable, contributed significantly to this view. This decision also seemed to call into question the patentability of other computer program innovations.[222] Two subsequent Supreme Court decisions, *Parker v. Flook* in 1978 and *Diamond v. Diehr* in 1981, reaffirmed the *Benson* ruling on the unpatentability of algorithms. Even though the Supreme Court did conclude that *Diehr's* invention was patentable, the *Diehr* decision was regarded as a very limited one for many years. It is limited in that it affirms only that patents can issue for traditionally patentable industrial processes which include a computer program as an element.

---

[220] Pamela Samuelson, *Benson Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions*, 39 EMORY L.J. 1025 (1990).

[221] According to Shepard's only one judicial opinion has cited Samuelson's article, our old friend Chief Judge Archer's dissent in *Alappat*. *See supra* Part III.A.1. The lack of judicial attention to Samuelson's article can largely be explained by the fact that since the creation of the Federal Circuit with its nearly exclusive jurisdiction over patent cases there have been few patent cases that have gotten to the Supreme Court and none involving software patents. *See supra* Part III.C.

[222] I am afraid that even Samuelson failed to grasp the fact that all computer programs are algorithms (although not all algorithms are computer programs).

> Despite the consistency in the Supreme Court's rulings against the patenting of algorithms, the Patent Office is now issuing patents for a wide variety of non-industrial computer program-related inventions and even seems to be issuing patents for computer program algorithms. Some patent lawyers argue that this change is consistent with *Diehr*, which they read to hold that only claims for "unapplied" algorithms are unpatentable. Professor Donald Chisum, the prominent patent scholar, attacks the Supreme Court case law more directly by calling for *Benson* to be overruled and for patent law to embrace the patentability of algorithms and other computer program-related inventions.
>
> The purpose of this article is to restate the case against patent protection for algorithms and many other computer program-related inventions in order to clarify the legal and public policy debates on this important subject. . . .[223]

The article contains an excellent history of the treatment of software patents by the courts before the decisions in *Alappat* and *State Street*. Unfortunately, however, the Federal Circuit paid no attention to Samuelson's arguments.[224]

There is, however, one passage in this article where Samuelson makes a critical mistake, arguing, in effect, that computers are texts and that computer programs are machines:

> . . . . Programs defy the conceptual molds provided by the traditional patent and copyright systems for understanding how to protect intellectual work. The computer scientist and mathematician John von Neumann realized that it would be possible to construct a computer that could store not only the data on which the computer was to operate, but also sets of instructions (computer programs) to guide its operations. By executing different sets of stored instructions, a general purpose digital computer could "become" different machines.[225] The von Neumann machine operates by processing symbolic representations (encoded instructions which prescribe the order in which the computer is to perform

---

[223] Samuelson, *supra* note 220, at 1028–30 (footnotes omitted).

[224] Except for Chief Judge Archer's dissent in *Alappat*. *See supra* Part III.A.1.

[225] It is fortunate that at this point Samuelson resorted to scare quotes, since the machine obviously remained the same machine even though it was performing different functions, just as a food processor remains the same machine whether it is being used to make bread dough at one time and chop liver at another.

> specified functions to accomplish a given task).[226] With von Neumann's discovery, machines could become writings, and writings could become machines. . . .[227]
>
> . . .
>
> Lawyers want to be able to make firm distinctions and to make legal rulings turn on these distinctions. However, even seemingly meaningful distinctions, like those between computer programs and computers or between programs and data, have a measure of artificiality to them. There is no fixed dividing line between computer programs and computers because anything that can be implemented in software can also be implemented in hardware.226 Professor Newell explains why there is no firm line between data and programs:
>
> > As anyone in computer science knows, the boundary between data and program—that is, what is data and what is procedure—is very fluid. In fact, . . . there is no principled distinction in terms of form or representation of which is which. What counts is the total body of knowledge represented somehow in the assembled symbolic expressions. This totality determines the ultimate behavior of the : machine.[228]
>
> To come to terms with the legal implications of this fluidity is no easy task.[229]

Samuelson was undoubtedly correct when she claimed that not being able to tell a computer—a patentable machine—from a program—an unpatentable writing—makes it difficult for lawyers to

---

[226] That amounts to saying, correctly, that a Von Neumann machine processes the symbolic representations of the instructions—that is, the symbolic representation of an algorithm—just as it processes other data. For more about "Von Neumann machines," *See infra* note 400.

[227] This claim, unless taken metaphorically, is simply wrong, for the kickable machine and the written program are always radically different types of things: machines are made of matter; programs consist of information. The idea that machines can become writings and that writings can become machines, if taken literally, leads to a dangerous muddle that allows proponents of software patents to argue that computer programs should be patentable like any other machine.

[228] That no distinction can be drawn between programs and data is not a cause for legal confusion for neither the software nor the other data that together determine the behavior of a computer are patentable under the holding in *Benson*.

[229] Samuelson, *supra* note 220, at 1128–31 (footnotes omitted).

determine the patentability of software[230] She was wrong, however, in claiming that it is difficult to tell a computer from a program; what she should have said is that it is easy for lawyers and legislators, who for the most part know little about computers and programs, to confuse the two and it is that unwarranted confusion that makes the issue raised in *Benson* appear to be difficult and to justify the—by now largely successful—efforts to get the Federal Circuit to "overrule" *Benson*.

But be that as it may, Samuelson ultimately came to the reasonable conclusion that the courts—even the Supreme Court—should not overrule *Benson*.

> This article asserts that there is simply too much at stake—not only for the computer software industry, but for the public at large—for such an important change in the subject matter boundaries of the patent system to be made by the courts; especially given the courts' befuddlement thus far about computer program innovations in patent cases.

And then Samuelson demonstrates that very little has changed in the arguments about whether software should be patentable, except that the Court of Customs and Patent Appeals has been replaced by the Federal Circuit.

> As reflected in the popular press, the current "controversy" over software patents has nothing whatsoever to do with the fine points of the court decisions, doctrinal analysis, and Patent Office practice on which this article has mainly concentrated. Rather, the issue has a more legislative tone, as though the patentability of computer programs and algorithms is an open issue about which it would be appropriate to ask if patents are "good" or "bad" for the industry and for society. Predictions that patents may be harmful to the software industry, computer science, mathematics, or society as a whole have been quite frequent, even from some of the most well-known people in the software and computer science fields.
>
> Patent lawyers tend to respond to such concerns by saying that, although this may be an interesting social policy issue, the reality is that the legal system has already established a

[230] And gives a lot of support to Chisum's argument that software should be as patentable as any other machine.

rule in favor of software patents. Court decisions have upheld the patentability of program-related inventions. There are many issued patents now, there will be many more in future years, and all are presumed to be valid. It would take congressional action to alter what has become the status quo, and that is not likely to happen.

This article argues that the legal foundation on which the current belief in the patentability of all program-related inventions rests is not as solid as many patent lawyers would like to believe. The CCPA decisions upholding software patents are hardly models of enlightened clarity and well-explicated principles. Indeed, the only principle which seems to have guided that court's deliberations over the years is one of upholding the patentability of as many program-related inventions as possible while still appearing to show some respect for the Supreme Court's decisions.[231] Both the CCPA decisions and the Patent Office's current more generous attitude toward software patents go beyond what the Supreme Court approved in the *Diehr* case. As a result, the legal debate over the patentability of computer program-related inventions may not be as settled as some patent lawyers claim.[232]

Although the author's study of the computer program patent cases has caused her to be critical of those decisions and of Professor Chisum's argument for overturning *Benson*, it is primarily because of the widespread concerns about the ill effects of patents from within the industry and the technical community that she has pursued this study.[233]

V. ARE SOFTWARE PATENTS NECESSARILY BAD THINGS?

As Samuelson pointed out[234] there were back in 1990 "[W]idespread concerns about the ill effects of patents from within the industry and the technical community," concerns that are far more widespread today. There are, however, also those who believe that software patents are a good thing, increasing the amount of innovation in the software field and increasing the well being of

---

[231] Of course, with the decisions in *Alappat* and *State Street* the Federal Circuit would stop showing any respect for the actual decisions of the Supreme Court in *Benson*, *Flook* and *Diehr*.

[232] That, of course, is a milder way of saying one of the things what I am claiming here.

[233] Samuelson, *supra* note 220, at 1133–34 (footnotes omitted; emphasis added).

[234] *See supra* text accompanying note 232.

society as a whole, while others, like Professor Chisum, believe that those who would exclude software from the processes that are treated as patentable subject matter must bear the burden of proof—and cannot satisfy it.[235]

Although, as I have argued—quite persuasively I think—that the law as established by the Supreme Court is that software is not patentable, that does not mean that the rulings of the Supreme Court on this subject should not be reversed. Although software is not patentable, perhaps it should be.

Now as far as I can tell no arguments were made by the Federal Circuit in *Alappat* and *State Street* as to why software should be patentable, other than the implicit claim that software should be patentable like any other process.[236]

As it turns out, however, there is a great deal of argument[237] about whether software patents—as distinguished from other types of patents—are good things or bad things from the point of view of encouraging innovation[238] or improving the economy.

Frankly, I doubt whether arguments that software patents increase or decrease innovation or our economic well-being can ever persuade anyone to change their mind about the such matters. And, in fact, I

---

[235] Over the years, there has been a lively debate among economists, lawyers and others over the economic and social justification for a patent system. Do the grants of limited exclusive rights really induce a higher level of innovation or a higher degree of utilization of innovations? Do such grants really induce disclosure of useful information that otherwise would not be disclosed? Is the incrementally higher level of innovation or disclosure worth the economic cost of higher prices and lower production that exclusive rights may cause? At times, the debate is framed in conceptual terms. For example, is a patent for an invention a "monopoly," which should be kept within bounds given this country's "traditional antipathy" toward monopolies, or is it "property," which should be afforded the same full legal protection as is afforded other types of tangible and intangible property?

Interesting though these questions about the patent system *in gross* may be, they need not and should not be raised anew or debated *de novo* in deciding whether algorithms constitute patentable subject matter. The patent system is a given. It was established by statute in 1790 and has been maintained by Congress ever since. This represents a policy determination by Congress that in fact the public interest is served by offering limited periods of exclusive rights in disclosed innovations in the useful arts. The statutory scheme includes within its subject matter the category of "processes" in a broad sense. In the *Flook* decision, the Supreme Court admitted that a mathematical algorithm came within the normal definition of a process. Given that literal inclusion of algorithms within the patent system, the burden of proof on the excludability of algorithms in a judicial or administrative forum should shift to the side that seeks such exclusion. The burden of proof should be carried only by a demonstration that (1) the evident policies of the statutory patent system prescribe that the word "process" be given a special nonliteral meaning that excludes algorithms or (2) the constitutional provision that authorizes Congress to establish a patent system places algorithms outside the permissible scope of such a system. Chisum, *supra* note 32 at 1010–11 (footnotes omitted).

[236] We shall see, hereafter that information processing is quite unlike processes that modify material substances. *See infra* Part VI.C.

[237] *See, e.g.*, Wikipedia, *Software Patent Debate*, http://en.wikipedia.org/wiki/Software_patent_debate.

[238] Which is what patents are supposed to do.

sincerely doubt that any of those arguments could possibly be "rational" in the sense that their results could be calculated by a properly programmed computer.[239]

The issue is confused by the fact that, even if software patents are not necessarily bad things, many of them—like the patent at issue in *State Street*—should never have been issued.

And there is also the problem that if special rules are to apply to software patents, why should there not also be special rules for other types of patents: one rule for patents for means of cooking rubber, another for patents for means of making chocolate cookies, and another for patents on adding machines, until the whole body of patent law becomes as balkanized, complex, and unworkable as the law of copyright.[240]

The later argument might be persuasive—and I would have to admit that it is not so easy to justify discrimination against software

---

[239] I find it amusing—and instructive—to imagine a patent on a computer program that proves that software patents increase—or, even more amusingly, decrease—the well being of society. I suppose that I should admit a certain bias here, for I have long thought, and have argued at length, that economic arguments—about much simpler issues than the economic consequences of software patents—are a complete waste of time. *See* Junger, *A Recipe for Bad Water: Welfare Economics and Nuisance Law Mixed Well*, 27 CASE W. RES. L. REV. 3 (1976). Thus fear that all those law review articles that describe patents as "non-rivalous goods," and discuss whether that means that they are not going to be allocated efficiently, are exercises in religious dialectics that bear no relation to any reality that we can actually experience. For a discussion of the economic issues relating to patents, see François Lèvéque & Yann Méniére, *The Economics of Patents and Copyright* (2004), http://www.bepress.com/cgi/viewcontent.cgi?article=1001&content=leveque. You might also look at ERICK STASIK, NOT SO PATENTLY OBVIOUS (2006), a book that is addressed primarily to law students. (I should hope that most law students would have the good sense not to take this treatise very seriously, while still realizing that it can supply arguments that may be found persuasive by their more gullible professors and perhaps even some judges.) For citations to many articles and reports on the economics of software patents, see *Research on the MacroEconomic Effects of Patents*, http://swpat.ffii.org /vreji/minra/sisku/.

[240] The Copyright Act has special provisions relating to:

1. anonymous works;
2. architectural works;
3. audiovisual works;
4. collective works;
5. compilations;
6. computer programs;
7. derivative works;
8. joint works;
9. literary works;
10. motion pictures;
11. phonorecords;
12. pictorial, graphic, and sculptural works;
13. pseudonymous works;
14. sound recordings; and,
15. transmission programs

as well as "food service or drinking establishments" and many other arcane matters. Copyright Act, 17 U.S.C. § 101 (2000).

patents—unless there is some reasonable basis for treating software inventions differently from other inventions. Fortunately for my claims in this article, there is indeed such a basis: permitting patents on means of processing information amounts—as Supreme Court held in *Benson*—to permitting the patenting of ideas, *i.e.*, the very processes of thought.[241]

Unlike patentable processes like cooking rubber—or cooking seagulls—or constructing adding machines, the processes specified in computer programs involve the processing of information, not of matter, and that makes all the difference. Computer programs are not tangible machines—or machines of any kind—despite what the Federal Circuit says in *Alappat*; computer programs are information about how one can use a computer—or one's head and one's hands—to process other information.[242]

A more lawyer-like, or, at least, lawyer-convincing, version of this argument is to point out that the constitutional authority for the granting of patents—and copyrights—is contained in Clause 8 of Section 8 of Article I of the United States Constitution, which provides:

> The Congress shall have Power . . . To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries.

What is critical for our purposes here it the distinction between "writings" that are copyrightable in order to promote the progress of "science" and "discoveries" that are patentable in order to promote the progress of the "useful arts." At the time of the writing of the Constitution and the first Patent Act in 1790 "science" referred to "knowledge" in general, not to the more limited range of subjects that we think of today when we hear the word "science."[243]

On the other hand,

> The men who participated in drafting the Constitution and the first patent act in the United States (Patent Act of 1790) conceived of a world in which the term

---

[241] *See infra* Part VI.B.6.

[242] Including other computer programs.

[243] "In the 17th and 18th c. the notion now usually expressed by science was commonly expressed by philosophy." OXFORD ENGLISH DICTIONARY ONLINE, *Science* ¶ 5.a. http://dicitionary.oed.com/cgi/entry/50215796?single=1&query_type=word&queryword=Science&first=1&max_to_show=10.

> "arts" captured within it various skills and branches of learning. The realm of the arts was divisible into cultural and useful arts, the former including liberal arts (grammar, logic/dialectics, rhetoric, arithmetic, geometry, music, and astronomy) and fine arts (painting, drawing, architecture, sculpture, poetry, dancing, and drama). . . . The useful arts, then, comprised the "mysteries" of the European craft guilds; these mysteries were processes and tools used by trained human beings operating in the physical world to produce physical objects generally regarded as being practically useful and marketable to other human beings.[244]

As we shall see, computer programs are texts—*i.e.*, writings—containing instructions on how to perform arithmetical and other mathematical and logical operations and, as such, may be copyrightable, but they are not within the useful arts, and thus are not patentable, because they do not "operate in the physical world to produce physical objects."

VI. OF COMPUTERS, SOFTWARE, AND INFORMATION

I am not unaware that there are conceptual difficulties in understanding, and justifying, the Supreme Court's decision in *Benson*, *Flook* and *Diehr*. That is, after all, my excuse for writing at such length about those decisions. There are, I am sure, many who would argue that the Federal Circuit was right in refusing to follow the Supreme Court's decisions in those cases as a matter of policy, if not of jurisdiction, and that if the issue were to come again before the Supreme Court,[245] the Court should overrule those cases.[246]

There is one glaring problem with that position, however, for the fact is that the Court was right in *Benson*, *Flook* and *Diehr* in holding that algorithms and mathematical formulae are the equivalent of ideas and that they also are the equivalents of "natural laws," all of which should be unpatentable.[247]

---

[244] Laura R. Ford, *Alchemy and Patentability: Technology, "Useful Arts," and the Chimerical Mind-Machine*, 42 CAL. W. L. REV. 49, 55–56 (2005) (footnotes omitted; emphasis added).

[245] As it inevitably will now that the Federal Circuit no longer has exclusive jurisdiction over appeals in patent cases. *See supra* text accompanying note 111.

[246] It would be a cruel sort of fun to hear the argument of counsel trying to persuade the Court that the Federal Circuit was right when it refused to follow the Court's decisions; I wonder whether anyone would actually dare make such an argument.

[247] *See supra* note 40 and accompanying text as well as the remainder of this article.

These are, I fear, truths that are difficult for lawyers to grasp. Most of us have—quite reasonably—an allergic reaction to discussions of the nature of "algorithms" or "ideas" or "mental processes" or "mathematical processes" or "information"—at least in the context of legal arguments. Who, after all, would want law cases resolved one way or the other because the judges of law or fact agreed or disagreed with Plato's concept of "Ideas" or Descartes's concept of the "mind"?

I submit, however, that we are never going to understand what was—and is—at issue in *Benson* and its progeny, if we do not understand the meaning of such terms in the context of computers and software. It is important, if we are to understand *Benson*, that we understand what computers are and what computer programs are and how, despite what many appear to think, these two terms are not interchangeable. Furthermore, if we are going to understand what computers are and what computer programs are we denizens of the so-called "information age" are going to have to at least begin to understand what information is.

And if we are going to understand why software should not be patentable we are going to have to cure ourselves once and for all of the perverse delusion that machines can be texts and that texts can be machines.[248] And we are also going to have to cure ourselves of the delusion that a process, like a thought, that processes only information, rather than matter or energy, is somehow more patentable than other abstract ideas and mental processes.[249]

In the remainder of this article I shall therefore be discussing some matters that are not typically discussed in law review articles.[250] I cannot, of course, set out a detailed history here of the evolution of computers nor can I give a detailed description of their nature and the nature of their software.[251] I hope, however, this following sketch will be sufficient to persuade you that programs are more like ideas than they are like machines.

If you already understand that a text cannot be a machine, or a machine a text, and that all that computers and their programs do is process information, just as we process information when we are doing computations with the aid of nothing but a pencil and some

---

[248] That delusion is the basis on which the Federal Circuit in *Alappat* held that software is patentable. *See supra* Part III.A.

[249] That delusion is the basis on which the Federal Circuit held in *State Street* that computations are patentable. *See supra* Part III.B.

[250] I compensate for this deviation from the standard by another deviation: I spend little time indulging in the nearly mandatory, and normally unilluminating, "economic analysis" of law—or of whatever. I figure that I can get away with it because my job description now includes the word "emeritus."

[251] I have neither the competency nor the room to do that.

paper, then you do not really need to read further in this section—but if you do you will may well find some of the material not uninteresting. If, on the other hand, you are not convinced of those facts, then you really should read on, although you may want to skip some of the more technical sounding parts.

For convenience I am next going to discuss computers—*i.e.*, the machines[252]—and computer programs—*i.e.*, descriptions of what those machines[253] are supposed to do—as if they were to some extent separable. In reality, of course, computers and their programs are inseparable when they are actually computing something:[254] the computer is a material device, the program is what that device does, is its functionality.[255] If computer programs are patentable subject matter, then novel melodies[256] should be patentable, even though they can be hummed by a human being, played upon a piccolo, or played mechanically by a player piano.[257]

A. Computers

The words "computer" and "computing" have a somewhat misleading connotation when applied to machines, since computation is usually thought of as an act that can be performed only by intelligences, and—for all practical purposes in so far as we know—only by human intelligences. In fact a "computer" was originally, as the Second Edition of the Oxford English Dictionary puts it: "One who computes; a calculator, reckoner; spec. a person employed to make calculations in an observatory, in surveying, etc."[258] It is rather as if we were to call a typewriter simply a "writer" and then assume that it—rather than a human author—actually does the writing. When a computer is, as it almost always is today, a machine rather than a human being, it would perhaps be better to call it something like a "tool for computation."

---

[252] Or people.

[253] Or people.

[254] "The computer poses fundamental challenges to traditional definitions of technology because it combines attributes of the human mind with those of a machine and because the end results of its activities inextricably combine the physical realm of hardware and the symbolic realm of software." Ford, *supra* note 244, at 52.

[255] Let me point out that not all functions are patentable; for example, surely no one would dream that the following "factorial function," though useful, could be patented if only it were novel,

$$\Gamma(z+1) = z!$$

for, were it patentable, every mathematical function would fall into the category of patentable subject matter.

[256] Which are useful for soothing the savage beast, etc.

[257] *Cf. supra* note 121.

[258] *See supra* notes 23 & 24 and accompanying text and *infra* Part VI.A.1.

To make matters worse, the type of machine that we call a "computer" is used for much more than merely computing, it is also used for word processing and data processing and language translation and any other process, such as encrypting and decrypting, that involves the manipulation of symbols—or, rather, of signs. It would probably abate some of the confusion surrounding computers were we to call them—as we sometimes do—"information processors."[259]

1. Computers as People

Once upon a time—during my life time, in fact—computers were people. When I was in high school[260] my teacher in the course in plain geometry worked one summer as a computer for my father, although, according to my father, he could not always compute. In those days, in the late forties of the last century, "computer" was an occupational title—a job description—just like, for example "gravity observer." Now, I never was really a computer myself, but the summer that I let my father talk me, against my better judgment, into working as a gravity observer on a seismic exploration crew up in the Northwest Territories of Canada near the confluence of the Liard and the Nahanni rivers—a land that was primarily muskeg, which is just a fancy word for swamp—I worked for a couple of evenings for a computer who, because of his Glaswegian accent, was inevitably known as "Scotty."

It was my job to go slogging through the swamp with a gravity meter on my back and, every so often, when I came across a stake with certain markings placed on it by the surveyors, I would unsling the gravity meter, place it on a tripod, and look through an eye piece with a vernier dial, twisting the dial until a little bubble, very much like the bubble in a carpenter's level, appeared in the very center of the eyepiece. At that point I could, by reading the numbers on the dial, supposedly determine the force of gravity acting at that particular location to the nearest thousandth of a gal or so.[261] As I recall, being rather a klutz, I never could reach quite that degree of accuracy.

---

[259] This usage would also make explicit the fact that computers process "information," not "matter" as patentable machines are required to do. *See supra* note 119.

[260] Which, by the way, was the same high school that Vice President Cheney later attended.

[261] A gal is, according to the on-line version of the American Heritage Dictionary of the English Language, "The centimeter-gram-second unit of acceleration, equal to one centimeter per second per second."

That hardly mattered though since finding what we were looking far hardly required that degree of accuracy and, in any case, there were other forces—especially tidal forces[262]—that messed up the data that I was collecting.

We, Scotty and I, were looking for buried river beds whose existence in turn was likely to mess up the seismic data that the rest of the crew was collecting. The whole purpose of the entire project was to make records of the reflections that were detected from underground strata when some dynamite was exploded that would allow seismologists like my father to locate and map underground structures where significant amounts of oil and gas might be trapped. So the ultimate goal was to find oil, but Scotty and I were merely collecting and manipulating data that would help others locate river beds that might be a source of noise in their own data.

I was supposed to collect the data; Scotty the computer was supposed to manipulate it. That is, after all, what computers are supposed to do: manipulate data. So I supplied the data to him, and someone—a seismologist—gave Scotty instructions as to how he was to process that data. But there were actually a couple of evenings that Scotty handed me a slide rule[263] and gave me some hasty instruction on how to use it in order to filter out the noise attributable to tidal forces and changes in temperature and things like that that messed up the data I had collected.[264]

Scotty just wanted me to recheck some computations that had already been done by someone else. But that allows me to claim with at least some legitimacy that I myself was a computer for a couple of evenings, or, at least, a computer's assistant. I'm not quite sure that I ever understood exactly what it was that I was doing, other than sliding scales around inside the slide rule, but that did produce numbers which satisfied Scotty.

Computers are not required—or even encouraged—to understand exactly what it is that they are doing.

Now those instructions that the seismologist gave to Scotty and the ones that Scotty gave to me were, by definition, instructions to a computer. And a set of instructions to a computer is exactly what is meant by the term "computer program."[265]

---

[262] Those tidal forces were, of course, also caused by gravity, but they were attributable to the mass and location of the moon, which were not what we were concerned with.

[263] *See* http://www.hpmuseum.org/sliderul.htm.

[264] It was possible to do that because every two hours I had to return to my the point where I had started from two hours before and take another reading, which allowed an estimate to be made as to how much drift there had been between each of the points where I had taken a reading.

[265] Or, at least, one of the meanings of that term. *See supra* note 19 and accompanying text.

Keep that in mind. The earliest computer programs were sets of instructions to human beings like Scotty and myself.

Of course, Scotty and my old geometry teacher were not the only nor the earliest computers. During the second world war there were a large number of computations that had to be made and most of the computers making those computations were women—typically working in teams assisted by mechanical calculators. Thus the *History of the Manhattan Project* of The Manhattan Project Heritage Preservation Association, Inc. recounts in the section entitled *Evolving from Calculators to Computers*[266] that:

Computers were people using desk calculators when Los Alamos began. . . .

> Early calculations relating to the diffusion of neutrons in a critical assembly of uranium were made by Eldred Nelson and Stanley Frankel, who were members of Robert Serber's group in the Radiation Laboratory at the University of California, Berkeley, in 1942. When they came to Los Alamos in the spring of 1943, they ordered the same sorts of machines that they had used in California: Marchant and Friden desk calculators[267] to make the calculations required in the design of nuclear weapons.
>
> To perform some of these repetitive calculations, a group of scientists' wives were recruited to form a central computing pool. . . .
>
> . . . .
>
> Dana Mitchell, whom Laboratory Director J. Robert Oppenheimer had recruited from Columbia University to oversee procurement for Los Alamos, recognized that the calculators were not adequate for the heavy computational chores and suggested the use of IBM punched-card machines. He had seen them used successfully by Wallace Eckert at Columbia to calculate the orbits of planets and persuaded Frankel and Nelson to order a complement of them.
>
> . . . .

---

[266] *See* http://www.childrenofthemanhattanproject.org/HISTORY/H-06c18.htm.

[267] *See infra* note 289 and accompanying text.

> The new IBM punched-card machines[268] were devoted to calculations to simulate implosion, and Metropolis and Feynman organized a race between them and the hand-computing group. "We set up a room with girls in it. Each one had a Marchant. But one was the multiplier, and another was the adder, and this one cubed, and all she did was cube this number and send it to the next one," said Feynmann. For one day, the hand computers kept up: "The only difference was that the IBM machines didn't get tired and could work three shifts. But the girls got tired after a while."
>
> . . . .
>
> One important fact to notice here is that, although the "girls," as Feynman called the computers, were human beings, they clearly did not have to know how the program that they were implementing worked or even what it was intended to do. Each of them just sat there throughout the whole shift either just multiplying pairs of numbers, or just cubing numbers, or doing something equally mechanical, over and over again, a process that undoubtedly was at least as mind numbing as working on an assembly line.
>
> Computers may in the early days have been people, but that does not mean that they knew why they were doing what they did. Whatever feelings of love and hate, attachment and revulsion, boredom and intellectual curiosity they may have had, to the extent that the "girls" were viewed purely as computers, they were as functional as a ditch digger or a backhoe. Of course they did get tired and sometimes made errors, but then, the vacuum tubes on early electronic computers also wore out with great regularity and when those tubes were tired, those computers too were prone to error.
>
> There was, of course, a purpose behind instructing those computers to carry out particular programs, but that purpose belonged to the scientists at Los Alamos, not to the computers themselves.

---

[268] *See infra* note 291 and accompanying text.

2. Computers As Machines

Today, of course, we no longer think of computers as being people like Scotty and myself or the "girls." Rather we think of computers as devices, as mechanisms, that—for the time being at least[269]—lack the characteristics of sentient beings.[270]

Nowadays computers are tangible, "physical" devices containing wires and switches, or their equivalents,[271] that can be used to perform various operations on a stream of digits in accordance with their programs, which are normally encoded as a string of digits.[272] Those streams and strings of digits can represent—we can understand those digits as symbolizing—almost anything, but for the purpose of understanding how our modern electronic digital computers work, it is easiest to treat those digits as a binary representation[273] of a number or a string of numbers.[274] We can then, by sending the computer's central processing unit a set of instructions in the form of a stream of digital numbers, cause the computer to perform mathematical or logical operations upon a string of digits encoding a set of signs or

---

[269] In science fiction there are many works that explore the possibility of computers becoming sentient; the prime example being HAL in STANLEY KUBRIC, 2001: A SPACE ODYSSEY (Metro-Goldwyn Mayer 1968).

[270] Even back in the days when "computer" was a job description, the ideal computer was a person who could suppress all human characteristics and simply function as a machine, just as the ideal human steel driver approximated a machine as closely as possible:

> John Henry said to his Captain,
> "A man ain't nothin' but a man,
> And before I'll let your steam drill beat me down,
> Die with the hammer in my hand,
> Die with the hammer in my hand."
> The steam drill, on the other hand, did not care whether it won or not. The steam drill did not care about anything.

[271] Some students at MIT once made out of a set of Tinker Toys a simple computer that was programmed to play tic-tac-toe. *See* http://www.rci.rutgers.edu/cfs/472 html/Intro/TinkertoyComputer/TinkerToy.html.

[272] *See infra* Part VI.A.3(b)(i). There is an important, if subtle, distinction, between a "stream" and a "string." A "stream" takes place in time, with one digit being sent to the computer at a time; a "string," on the other hand, occupies space and is all present at the same time, like the string of digits

1 2 3 5 7 11 13

that appears before you as you read this article in a journal or a computer screen. On the other hand, if someone were to read the text of that string to you out loud, you would apprehend it as a stream. The stream is an on-going process; the string is a set of symbols, *i.e.*, a text.

[273] Recall that in *Gottschalk v. Benson*, where the Supreme Court first held that a computer program was not patentable, the "invention" at issue was a process for converting binary coded decimal representations of numbers into simpler and more useful binary representations.

[274] What a "number" is an interesting and contentious metaphysical question that need not concern us here. *See infra* note 280.

symbols and thus produce another string or stream of digits that we can interpret as representing other signs or symbols.

Or something like that.

A simpler definition is that a computer is a device that can be used to process information.[275]

But let us look a little more deeply into this matter.[276]

3. Protocomputers: Tools to Assist in Calculations

There have been devices, if we are willing to count fingers as devices, that we have used to assist us in calculating since the first members of our species walked upon the earth. Fingers are still used today as memory devices to help us remember how far we have counted, and counting is the basic arithmetic operation. The fact that we have ten fingers—ten digits—on our hands helps to explain why today we commonly use a ten-based numbering system, rather than a binary or octal or duodecimal system.

Fingers are also useful when performing the arithmetic function of addition. If one set of items has five members and one has three and we represent each member by a finger, then all we have to do if we want to calculate how many members there are in the two sets is to count the five fingers that represent the members of one set and then the three fingers that represent the members of the other set.[277]

Now a major problem with using fingers for calculations is that we don't have enough fingers to count very far, although we can stretch their capacity a bit by letting each finger stand let us say for ten items.

---

[275] What information is a difficult question, but one that is of critical importance to an understanding of the nature of computer programs. *See infra* Part VI.C.

[276] For an overview of the historical development of devices that assist in calculating and otherwise processing information you may want to look at the "time line" included in Jeremy M. Norman, *From Gutenberg to the Internet: An Annotated Chronology of the History of Information from About 30,000 B.C.E. to the Present*, http://www.normanpublishing.com/G2I/docs/timeline/index.shtml. Using references to discoveries, social developments, and documents, this timeline attempts to arrange in approximate chronological sequence landmarks in the history of methods used to record, distribute, exchange, organize, store, and search information. Topics include Writing, Manuscript Copying, Papermaking, Printing, Publishing, Bibliography, Libraries, Archives, Survival of Information, Book and Manuscript Collecting, Conservation, Education, Computing Theory, Computing, Software, Networking, Telegraph, Photography, Cinematography, Telephone, Radio, Television, Cryptography, Computer Games, Law, Privacy, and related fields. One of ways that the timeline attempts to reflect the "survival of information" is to discuss the earliest extant texts of documents, and in certain instances the history of their survival in physical form. The timeline also tracks projects for the long term preservation and conservation of digital objects. *Id.*

[277] The total number of fingers representing the members of the two sets is, of course, eight.

A simple solution to that problem that probably occurred early in the history of our species is to use some small objects, such as pebbles, so that each pebble represents an item in the set of things to be counted.[278] In time the pebbles came to be used with counting boards, which were the forerunners of the abacus,[279] and the abacus itself, in which beads on wires replace the pebbles on a counting board, still is used in some countries to perform calculations.[280]

In this list of primitive tools for calculation we should probably include a very different sort of device: the portions of the human brain that are capable of doing primitive and limited calculations without any additional tools.

> Number is a fundamental parameter by which we make sense of the world surrounding us. Not only can we quickly and accurately perceive the numerosity of small collections of things; but all languages have number words; all of us have learned, more or less spontaneously, to calculate on our fingers; and most of us have strong arithmetic intuitions which allow us to quickly decide that 9 is larger than 5, that 3 falls in the middle of 2 and 4, or that 12+15 cannot equal 96, without much introspection as to how we perform those feats. I collectively refer to those fundamental elementary abilities or intuitions about numbers as "the number sense"
>
> My hypothesis is that number sense qualifies as a biologically determined category of knowledge. I propose that the foundations of arithmetic lie in our ability to mentally represent and manipulate numerosities on a mental "number line", an analogical representation of number; and that this representation has a long evolutionary history and a specific cerebral substrate. "Number appears as one of the fundamental dimensions according to which our nervous system parses the external world. Just as we cannot avoid seeing objects in color . . . , in the same way numerical quantities are imposed on us effortlessly through the specialized circuits of our inferior parietal lobe. The structure

---

[278] The Latin word for pebble is calculus which is the root of the English words "calculate" and "calculator" and, of course, "calculus."

[279] *See* http://www.mathmojo.com/abacus/abax/abax2.html.

[280] In Japan in 1946 a contest was held to see which was speedier, an abacus or a modern—at that time—electric calculator. The abacus won. *The Abacus vs. the Electric Calculator*, http://www.ee.ryerson.ca/ elf/abacus/abacus-contest.html.

> of our brain defines the categories according to which we apprehend the world through mathematics."[281]

Perhaps the most that can be said about such matters is to ask the question first raised by Warren McCulloch: "What is Number, that a Man may know it, and a Man, that he may know a Number?"[282]

In more modern times more complex mechanical calculating machines were invented, although they, like the abacus, still had to be programmed by head and hand, a human being turning dials or depressing keys. They were programmed to perform various numerical calculations, but it was still a human being who had to implement the program.

In 1649, Blaise Pascal received a patent for his calculating machine[283] from King Louis XIV of France.

> Pascal invented the first digital calculator to help his father with his work collecting taxes. He worked on it for three years between 1642 and 1645. The device, called the Pascaline, resembled a mechanical calculator of the 1940s. This, almost certainly, makes Pascal the second person to invent a mechanical calculator for Schickard[284] had manufactured one in 1624.[285]

> There were problems faced by Pascal in the design of the calculator which were due to the design of the French currency at that time. There were 20 sols in a livre and 12 deniers in a sol. The system remained in France until 1799 but in Britain a system with similar multiples lasted until 1971. Pascal had to solve much harder technical problems to work with this division of the livre into 240 than he would have had if the division had been 100. However production of the machines started in 1642 [and by] 1652 fifty prototypes had been produced, but few machines were sold, and

---

[281] Stanislas Dehaene, *Précis of "The number sense"*, http://www.unicog.org/publications/Dehaene PrecisNumberSense.pdf.

[282] WILLIAM MCCULLOUGH, EMBODIMENTS OF MIND. (Cambridge, MA: The MIT Press 1963).

[283] Note that this patent covered the tangible machine not the instructions on how to use it to make calculations.

[284] *See 1625 AD: Wilhelm Schickard's Mechanical Calculator*, http://www.maxmon.com/1625ad.htm.

[285] Shickard described his machine in a letter to the astronomer Johannes Kepler, but no examples of the machine have survived, although prototypes have been built based on his description. *Id.*

> manufacture of Pascal's arithmetical calculator ceased in that year.[286]

Gottfried Wilhelm Leibnitz, who invented the differential calculus[287] at about the same time as Isaac Newton, designed a greatly improved version of Pascal's calculator around 1673 called the "Stepped Reckoner," but it was not until the early 1800s that

> Charles Xavier Thomas de Colmar (1785-1870) made the first practically successful calculator, the Arithmometer. This worked on the stepped gear principle of Leibnitz and could perform all four basic arithmetic operations. . . .
>
> The gear train of the arithmometer was driven in the forward direction only and could perform addition and multiplication. To perform subtraction or division a reversing gear had to be engaged.
>
> The first patent for this device was granted in November 1820. Derivative devices were still being manufactured after 1900 and some were still in use in the 1940s.[288]

By the 1940's many different types mechanical calculators were being marketed and electric calculators had been introduced that did not require the operator to turn a crank in order to cause the calculator to carry out its operations. Among these were the Marchant and Friden Calculators that were used by the human computers of the Manhattan Project.[289] These computers were "programmed" by depressing the keys on a keyboard and they were capable of doing the basic mathematical operations of addition, subtraction, multiplication, and division, although they did so quite noisily.[290]

Another type of electric powered calculating machine was the electric tabulating machine designed by Hermann Hollerith to assist in calculation the results of the 1890 census.[291] Descendants of this

---

[286] http://www-groups.dcs.st-and.ac.uk/ history/Mathematicians/Pascal.html.

[287] Do you think that anyone at that time would have believed that Leibnitz—or his competitor Newton—could have been granted a patent on the differential calculus? Differential calculus was classified as belonging to science—or philosophy—not to the useful arts.

[288] *Old Computer Hut: In the Beginn*ing, http://tinyurl.com/sxqlk.

[289] *See supra* note 267.

[290] I remember instructing one of these calculators—one that was bolted to a table with metal legs—to divide a positive integer by zero: the calculator shook so energetically that the table started walking across the floor. The only way to stop it that I ever found was to pull the plug out of the wall socket.

[291] H. Hollerith, *An Electric Tabulating System*, THE QUARTERLY 238–55, Columbia University School of Mines, Vol. X, No. 16 (Apr. 1889).

tabulator were also used in the Manhattan Project.[292] Data was fed into the tabulator using punch cards, which were first used to control the Jacquard loom[293] and later were used to feed programs and data into early mainframe computers.[294] The original electric tabulators were hard wired and could not easily be reprogrammed; later models had a plugboard which allowed them to be reprogrammed the same way that the ENIAC was reprogrammed.[295]

a. Two Computers That Never Were

Before the first actual computer in the modern sense was built, there were two prototypes that were never constructed. One was not constructed because it was a mechanical device whose components needed to be made with an accuracy that probably could not even be duplicated today, to say nothing of the time when it was designed; the other because it was conceived of as part of a thought experiment and thus was never intended to actually be built.

(i) *Babbage's Analytical Engine*

More than a hundred years before the first working digital computers were actually constructed one was designed by Charles Babbage:

> Seldom, if ever, in the history of technology has so long an interval separated the invention of a device and its realization in hardware as that which elapsed between Charles Babbage's description, in 1837, of the Analytical Engine, a mechanical digital computer which, viewed with the benefit of a century and a half's hindsight, anticipated virtually every aspect of present-day computers. Charles Babbage (1792-1871) was an eminent figure in his day, elected Lucasian Professor of Mathematics at Cambridge in 1828 (the same Chair held by Newton and, in our days, Stephen Hawking); he resigned this professorship in 1839 to devote his full attention to the Analytical Engine. Babbage was a Fellow of the Royal Society and co-founder of the British Association for the Advancement of Science, the Royal Astronomical Society, and the Statistical Society of London. He was a close

---

[292] *See supra* note 267.

[293] *See infra* Part VI.A.3(b)(ii).

[294] For a wonderful description of how punch cards were used to program mainframe computers back in the 1970's, *see* Dale Fisk, Programming *with Punched Cards*, http://www.columbia.edu/acis/history/fisk.pdf.

[295] *See infra* Part VI.A.4.

> acquaintance of Charles Darwin, Sir John Herschel, Laplace, and Alexander Humboldt, and was author of more than eighty papers and books on a broad variety of topics.
>
> His vision of a massive brass, steam-powered, general-purpose, mechanical computer inspired some of the great minds of the nineteenth century but failed to persuade any backer to provide the funds to actually construct it. It was only after the first electromechanical and later, electronic computers had been built in the twentieth century, that designers of those machines discovered the extent to which Babbage had anticipated almost every aspect of their work.[296]

Here is a contemporary description of the Analytical Engine:

> Those labours which belong to the various branches of the mathematical sciences, although on first consideration they seem to be the exclusive province of intellect, may, nevertheless, be divided into two distinct sections; one of which may be called the mechanical, because it is subjected to precise and invariable laws, that are capable of being expressed by means of the operations of matter; while the other, demanding the intervention of reasoning, belongs more specially to the domain of the understanding. This admitted, we may propose to execute, by means of machinery, the mechanical branch of these labours, reserving for pure intellect that which depends on the reasoning faculties. Thus the rigid exactness of those laws which regulate numerical calculations must frequently have suggested the employment of material instruments, either for executing the whole of such calculations or for abridging them; and thence have arisen several inventions having this object in view, but which have in general but partially attained it. For instance, the much-admired machine of Pascal is now simply an object of curiosity, which, whilst it displays the powerful intellect of its inventor, is yet of little utility in itself. Its powers extended no further than the execution of the first four operations of arithmetic, and indeed were in reality confined to that of the first two, since multiplication and division were the result of a series of additions and subtractions. The chief drawback hitherto on most of such machines is, that they require the

---

[296] John Walker, *The Analytical Engine: The First Computer*, http://www.fourmilab.ch/babbage/.

> continual intervention of a human agent to regulate their movements, and thence arises a source of errors; so that, if their use has not become general for large numerical calculations, it is because they have not in fact resolved the double problem which the question presents, that of *correctness* in the results, united with *economy* of time.
>
> Struck with similar reflections, Mr. Babbage has devoted some years to the realization of a gigantic idea. He proposed to himself nothing less than the construction of a machine capable of executing not merely arithmetical calculations, but even all those of analysis, if their laws are known. The imagination is at first astounded at the idea of such an undertaking; but the more calm reflection we bestow on it, the less impossible does success appear, and it is felt that it may depend on the discovery of some principle so general, that, if applied to machinery, the latter may be capable of mechanically translating the operations which may be indicated to it by algebraical notation. . . .[297]

And here is a modern description of Babbage's Analytical Engine:

> . . . . The Analytical Engine was not only automatic but also general purpose *i.e.* it could be 'programmed' by the user to execute a repertoire instructions in any required order. The engine was envisaged as a universal machine for finding the value of almost any algebraic function. The Analytical Engine is not a single physical machine but a succession of designs that Babbage refined until his death in 1871.
>
> The designs for the Analytical Engine include almost all the essential logical features of a modern electronic digital computer. The engine was programmable using punched cards.[298] It had a 'store' where numbers and intermediate results could be held and a separate 'mill' where the arithmetic processing was performed. The separation of the 'store' (memory) and 'mill' (central processor) is a fundamental feature of the internal organisation of modern computers.

---

[297] L.F. Menabrea of Turin, Officer of the Military Engineers, *Sketch of The Analytical Engine Invented by Charles Babbage*, from the Bibliothque Universelle de Genve, October, 1842, No. 82, *available at* http://www.fourmilab.ch/babbage/sketch.html.

[298] *See infra* note 291.

> The Analytical Engine could have 'looped' (repeat the same sequence of operations a predetermined number of times) and was capable of conditional branching (IF. . . THEN. . . statements) *i.e.* automatically take alternative courses of action depending on the result of a calculation.
>
> The Engine would have been vast. Had it been built it would have needed to be operated by a steam engine of some kind. Babbage made little attempt to raise funds to build the Analytical Engine. Instead he continued to work on simpler and cheaper methods of manufacturing parts and built a small trial model which was under construction at the time of his death.
>
> The movement to automate mathematical calculation in the nineteenth century failed and the impetus to continue this work was largely lost with Babbage's death. From the vantage point of the modern computer age we are better placed to appreciate the full extent to which Babbage was indeed the first pioneer of computing.[299]

Although the Analytical Engine was never completed it inspired The Right Honourable Augusta Ada, Countess of Lovelace, Lord Byron's only legitimate daughter, to write the first computer program—a program for calculating Bernoulli numbers.[300]

Although the Analytical Engine was never actually constructed, it has been "emulated"[301] on modern digital computers.

Now there surely is no question that Babbage could have obtained a patent on the Analytical engine, if it ever had been built. The question that we are concerned with here is: Could Lady Ada have obtained a patent on her program, a patent on the instructions that she wrote describing how to calculate Bernouli numbers on that machine?

---

[299] Science Museum, *Analytical Engine*, http://www.sciencemuseum.org.uk/online/babbage/page5.asp.

[300] The Bernoulli numbers may . . . be defined using the technique of generating functions. Their exponential generating function is x/(ex − 1), so that:

$$\frac{x}{e^x - 1} = \sum_{n=0}^{\inf} B_n \frac{x^n}{n!}$$

for all values of x of absolute value less than 2_ (the radius of convergence of this power series). Wikipedia, *Bernouli number*, http://en.wikipedia.org/wiki/Bernoulli number. *Nota bene*: I do not claim that I understand this definition. To view that first computer program, see http://psychclassics.yorku.ca/Lovelace/lovelace.htm.

[301] For what it means to "emulate" a computer, *see infra* note 315.

(ii) *Turing Machines*

Turing machines are imaginary devices that, unlike Babbage's Analytical Engine, were never actually intended to be built. On the other hand, Turing machines are capable, in theory at least, of running any program that can be run by any other computer, from human beings like Scotty,[302] through the Analytical engine, were it ever to be built, to the latest general purpose digital computer. In other words, there are Turing machines that can "emulate"[303] any computer whatsoever.[304]

We shall spend a little more time examining Turing machines and what is known as the Church-Turing Thesis, but for our purposes the critical fact about Turing Machines is that they demonstrate that programming a computer to perform a particular task cannot be viewed as the flipping of switches that thus create a new—and potentially patentable—machine, for the new program can equally well[305] be run on another computer that lacks the corresponding physical switches, or that like Scotty or the Turing machines imagined by Turing does not have any switches at all.

Turing machines were first imagined by Alan Turing[306] around 1936 in order to explore the limitations of any effective device that can be "programmed," as we now term it, to perform any logical or arithmetical operation. A Turing machine can thus perform the same operations that are performed by a modern general purpose computer. Turing machines were conceived—before the actual development of the programmable general purpose computer—as a way of exploring mathematically what is and what is not computable in theory.

The best description that I know of Turing machines and what they can and cannot do is given in the Stanford Encyclopedia of Philosophy in the entry entitled "Turing Machines:"[307]

---

[302] *See supra* Part VI.A.1.

[303] "To imitate the function of (another system), as by modifications to hardware or software that allow the imitating system to accept the same data, execute the same programs, and achieve the same results as the imitated system." The Free Dictionary, *Emulate*, http://www.thefreedictionary.com/emulate. *See infra* note 315.

[304] And all general purpose computers, including human computes, can emulate such Turing machines.

[305] Though perhaps not very efficiently.

[306] On computable numbers, with an application to the Entscheidungsproblem, http://www.abelard.org/turpap2/tp2-ie.asp. For biographical information about Turing, see The Alan Turing Home Page, Maintained by Andrew Hodges, author of *Alan Turing: the Enigma*, http://www.turing.org.uk/turing/, which describes him as: "Founder of computer science, mathematician, philosopher, codebreaker, strange visionary and a gay man before his time."

[307] http://plato.stanford.edu/entries/turing-machine/.

> Turing machines, first described by Alan Turing in (Turing 1937), are simple abstract computational devices intended to help investigate the extent and limitations of what can be computed.
>
> Turing, writing before the invention of the modern digital computer, was interested in the question of what it means to be computable. Intuitively a task is computable if one can specify a sequence of instructions which when followed will result in the completion of the task. Such a set of instructions is called an effective procedure, or algorithm,[308] for the task.[309] This intuition must be made precise by defining the capabilities of the device that is to carry out the instructions. Devices with different capabilities may be able to complete different instruction sets, and therefore may result in different classes of computable tasks . . . .
>
> Turing proposed a class of devices that came to be known as Turing machines. These devices lead to a formal notion of computation that we will call Turing-computability.
>
> The proposition that Turing's notion captures exactly the intuitive idea of effective procedure is called the Church-Turing thesis. This proposition, being a claim about the relationship between a formal concept and intuition, is not provable, though it would be refuted by an intuitively acceptable algorithm for a task that is not Turing-computable. That no such counterexample has been found, together with the fact that Turing-computability is equivalent to independently defined notions of computability based on alternative foundations, such as recursive functions[310] and abacus machines, indicates that there is at least something natural about this notion of computability.
>
> Turing machines are not physical objects but mathematical ones. We require neither soldering irons nor silicon chips to build one. The architecture is simply described, and the

---

[308] For more about algorithms, *see infra* Part VI.B.1. It should be recalled that the claimed invention in *Gottschalk v. Benson* was described by the Court as an "algorithm."

[309] It is worth noting here that a "sequence of instructions"—at least if designed to be carried out by a computer—is also exactly what today we call a "computer program."

[310] If you want to explore recursive functions and related matters—and even you do not think that you want to do that—I recommend that you at least look at Douglas Hofstadter's Gödel, Escher, Bach (1979).

> actions that may be carried out by the machine are simple and unambiguously specified. . . .
>
> . . . .
>
> The formal concept proposed by Turing is that of computability by Turing machine. He argued for the claim . . . that whenever there is an effective method for obtaining the values of a mathematical function, the function can be computed by a Turing machine. The converse claim is easily established, for a Turing machine program is itself a specification of an effective method: without exercising any ingenuity or insight, a human being can work through the instructions in the program and carry out the required operations.[311] If Turing's thesis is correct, then talk about the existence and nonexistence of effective methods can be replaced throughout mathematics and logic by talk about the existence or non-existence of Turing machine programs.[312]

The idea of the Turing machine underlies the Church-Turing Thesis that every effective computation can be carried out by a Turing machine—or its real-world equivalent, a general purpose digital computer.

> The Church-Turing thesis concerns the notion of an effective or mechanical method in logic and mathematics. "Effective" and its synonym "mechanical" are terms of art in these disciplines: they do not carry their everyday meaning. A method, or procedure, M, for achieving some desired result is called "effective" or "mechanical" just in case
>
> > 1. M is set out in terms of a finite number of exact instructions (each instruction being expressed by means of a finite number of symbols);
> >
> > 2. M will, if carried out without error, produce the desired result in a finite number of steps;

---

[311] In other words, a human being is the equivalent of a Turing machine.

[312] http://plato.stanford.edu/entries/turing-machine/. Thus the issue that we are discussing is whether Turing machine programs can be patented, programs that can be implemented by a human being? For a description of a Turing Machine, see Jonathan Steidel's, *What is a Turing Machine?*, http://www.science.gmu.edu/_jsteidel/801-prj/turing.html.

> 3. M can (in practice or in principle) be carried out by a human being unaided by any machinery save paper and pencil;
>
> 4. M demands no insight or ingenuity on the part of the human being carrying it out.[313]

Although Turing never intended that Turing machines should actually be constructed,[314] they have often been "emulated"[315]

---

[313] Stanford Encyclopedia of Philosophy, *The Church-Turing Thesis*, http://plato.stanford.edu/entries/church-turing/. Note that "effective" or "mechanical" methods are much like algorithms. *See infra* Part VI.B.1.

[314] A Turing machine is a kind of state machine. At any time the machine is in any one of a finite number of states. Instructions for a Turing machine consist in specified conditions under which the machine will transition between one state and another.

> A Turing machine has an infinite one-dimensional tape divided into cells. Traditionally we think of the tape as being horizontal with the cells arranged in a left-right orientation. The tape has one end, at the left say, and stretches infinitely far to the right. Each cell is able to contain one symbol, either '0' or '1'.
>
> The machine has a read-write head, which at any time scanning a single cell on the tape. This read-write head can move left and right along the tape to scan successive cells.
>
> The action of a Turing machine is determined completely by (1) the current state of the machine (2) the symbol in the cell currently being scanned by the head and (3) a table of transition rules, which serve as the ~201cprogram~201d for the machine.
>
> Each transition rule is a 4-tuple:
>
> <State0, Symbol, Statenext, Action>
>
> which can be read as saying "if the machine is in state State0 and the current cell contains Symbol then move into state Statenext taking Action". The actions available to a Turing machine are either to write a symbol on the tape in the current cell . . . , or to move the head one cell to the left or right . . . .
>
> If the machine reaches a situation in which there is not exactly one transition rule specified, *i.e.*, none or more than one, then the machine halts.
>
> In modern terms, the tape serves as the memory of the machine, while the readwrite head is the memory bus through which data is accessed (and updated) by the machine. There are two important things to notice about the definition. The first is that the machine's tape is infinite in length, corresponding to an assumption that the memory of the machine is infinite. The second is similar in nature, but not explicit in the definition of the machine, namely that a function will be Turing-computable if there exists a set of instructions that will result in the machine computing the function regardless of the amount of time it takes. One can think of this as assuming the availability of infinite time to complete the computation.
>
> These two assumptions are intended to ensure that the definition of computation that results is not too narrow. This is, it ensures that no computable function will fail to

> by programs running on other types of general purpose computers[316] and a few actual working physical models have been built—at least one out of Lego blocks.[317]

When one considers the patentability of computer programs it is significant that any computer program that can be run on a Turing machine or on a general purpose computer can be run—if perhaps not very efficiently—on any other general purpose computer[318] or any universal Turing machine. This means that machines that are capable of carrying out any computable computation already exist, just waiting perform any calculation that may be specified by any mathematical algorithm, *i.e.*, any computer program whatsoever.

b. Devices for Storing Information

Most of the devices that we have considered up to now that are used to assist us in computation are to a greater or lesser extent what might be called gadgets, that is, they have moving parts that have

---

> be Turing-computable solely because there is insufficient time or memory to complete the computation. If a function is not Turing-computable it is because Turing machines lack the computational machinery to carry it out, not because of a lack of spatio-temporal resources.

Stanford Encyclopedia of Philosophy, *Turing Machines*, Description.

[315] The emulators, of course, do not have an infinite memory.

> A software emulator allows computer programs to run on a platform (computer architecture and/or operating system) other than the one for which they were originally written. Unlike simulation, which only attempts to reproduce a program's behavior, emulation attempts to model to various degrees the state of the device being emulated. High-level emulation uses a combination of the two approaches in an attempt to retain as much accuracy as possible while having the advantages of simplicity and speed provided by simulation.
>
> ....
>
> In a theoretical sense, the Church-Turing thesis implies that any operating environment can be emulated within any other. In practice, it can be quite difficult, particularly when the exact behavior of the system to be emulated is not documented and has to be deduced through reverse engineering. It also says nothing about timing constraints; if the emulator does not perform as quickly as the original hardware, the emulated software may run much more slowly than it would have on the original hardware, or it may run too fast to be usable.

Wikipedia, *Emulator*, http://en.wikipedia.org/wiki/Emulator. Note that any emulator must ultimately run on hardware of some sort even though the program emulated is emulated in software.

[316] http://www.comphist.org/HistoricMachines/TM/Index.html.

[317] http://http://tinyurl.com/d3tc3.

[318] Including human beings.

some sort of mechanical function. One possible exception is the counting board and pebbles mentioned in Part VI.A.3, and even in that case one could argue that those proto-abacuses do have moving parts and thus fall within the definition of a "gadget."

Our major difficulty, however, when doing computing in our heads is remembering what the results are that we have calculated and remembering what it is that we still have to do in the future. That is why in defining an effective method for performing calculations it is stipulated that the method can be carried out by a human being aided by no devices except paper and pencil.[319] Since calculations can be broken down into a series of simple, "mechanical" steps, the critical need in doing computations is not for gadgets that can carry out those steps—which we can, after all, do in our heads—but some devices like pencil and paper—or fingers or pebbles—that can assist us in remembering.

In other words, some means of storing information[320]—the information to be processed and the information that has been processed—is essential for all computers, whether human or machine. The ability to do computations is of secondary importance.

This is amusingly illustrated by the fact that between 1959 and 1970 IBM made a computer—the 1620—that did addition and subtraction by looking up the answer in a table stored in "memory" rather than actually doing calculations. This machine was popularly known as the "CADET," an acronym standing for "Can't Add, Doesn't Even Try."[321]

The invention of the printing press with movable type by Johannes Gutenberg in 1536 can be viewed as a major contribution to the series of devices that have been used to assist in making computations, for a computation is of no value to anyone unless there is someway to store its result. Devices that can process information but cannot store it are pretty much worthless.

The development of "memory devices" in which information can be stored is a critical part of the evolution of modern computers,[322] whether men[323] or machines. Thus when I was in high school studying

---

[319] *See supra* text accompanying note 314. Recall that in *Benson* Justice Douglas stressed the fact that the algorithm at issue there could be carried out "by head and hand." *See supra* note 76 and accompanying text.

[320] As to what information is *See infra* Part VI.C.

[321] The IBM 1620 Data Processing System, http://tinyurl.com/ddxxk.

[322] Digital computers have two key components, the central processing unit where the actual processing is done—which was called the "mill" in the plan of Babbage's Analytical Engine—and the memory where the input data and the results of the computations are stored—which was called the "store" by Babbage.

[323] Or women.

algebra, one of the things that we had to learn how to do in making computations was to look up values in the tables of logarithms that were contained in books like the Standard Mathematical Tables[324] published by the Chemical Rubber Company.[325] Today, of course, we just have a computer compute the values of logarithms for us, as we need them.[326] Thus modern computers and computer programs have to a large extent replaced books and other more static information storage devices that were once used in making computations by head and hand and have also replaced those primitive storage devices with memory chips and other electronic and optical devices.

That suggests, of course, that if the contents of books like Standard Numerical Tables and Formulae are not patentable—and, as far as I know, no one suggests that they are—then computer programs that produce the same information on demand should be equally unpatentable. There is, however, some danger that, if *Alappat*[327] is still considered to be a good precedent, a memory device like a CD-ROM—or a player piano role—that is storing numerical tables and formulae would be patentable as a "manufacture" because it would be embodied in a physical structure.[328]

(i) *Characters, Numerals, & Other Symbols*

Books and other devices for the storage of information would have been worthless if there had not been the earlier invention of characters—or, more precisely, glyphs[329]—that enable a written

---

[324] *See* http://home.comcast.net/~lkrakauer/CRC99ph/CRCbook.htm. The Standard Numerical Tables and Formulae are still available: http://www.mathnetbase.com/ejournals/books/book summary/summary.asp?id=1129.

[325] Now the CRC Press.

[326] And, probably, we also have the computer do the entire series of calculations for which we needed the value of the logarithms.

[327] *See supra* Part III.A.

[328] *See supra* note 123 and accompanying text.

[329] [G]lyph In information technology, a glyph (pronounced GLIHF; from a Greek word meaning carving) is a graphic symbol that provides the appearance or form for a character. A glyph can be an alphabetic or numeric font or some other symbol that pictures an encoded character. The following is from a document written as background for the Unicode character set standard. An ideal characterization of characters and glyphs and their relationship may be stated as follows:

> 1. A character conveys distinctions in meaning or sounds. A character has no intrinsic appearance.
>
> 2. A glyph conveys distinctions in form. A glyph has no intrinsic meaning.
>
> 3. One or more characters may be depicted by one or more glyph representations (instances of an abstract glyph) in a possibly context dependent fashion. In the

representation of the sounds and concepts of our spoken languages, the primary means by which we communicate information. The invention of such characters—of which the Roman alphabet and the Arabic numerals are for most of us the most familiar examples—depended, in turn, upon the "invention" of those languages—and of language itself. "Invention" is almost certainly the wrong word to use here since language—the ability to acquire and speak a language, or several languages—seems to have evolved as the defining characteristic of our species. Obviously no one—not even Noam Chomsky[330]—could ever have obtained a patent on that "means or process of conveying information"[331] that we know as language. On the other hand, if the Federal Circuit were correct in its holdings in *Alappat* and States Street, what would prevent someone from getting a patent on a newly created language such as Esperanto or Interlingua? Can you imagine a legal regime where one would be required to pay a royalty every time one thought a thought in Interlingua or expressed it in that tongue? Or one where you could be enjoined from thinking in that language? Under a legal regime where the means of thought can be patented, only the thoughtless will be free.

A fundamental feature of all oral languages[332] is that they are processes. Speech is a process, something that one does and that then vanishes without a trace unless it is stored in some sort of memory. Writing is also a process just like speech, but in the case of writing the process also produces a static copy—a written record or written text—that can be read at some other time and some other place. It is the paper or papyrus or clay tablet, the punch card or CD or memory chip, on which the writing is stored—*i.e.*, that memory device—and the tools like pens and pencils and styluses and paper punches that are used in writing on those media that concern us here. An improvement

---

Unicode standard, a character is stated to be an abstract entity and not a glyph (some visual representation of a character).

This definition is taken from the Whatis on-line data base, http://whatis.techtarget.com/definition/0,,sid9 gci212200,00.html.

[330] For information about Chomsky, the founder of modern linguistics, see Chomsky.info, http://www.chomsky.info/. For an introduction to his linguistic theories, see NOAM CHOMSKY, SYSTEMS OF SYNTACTIC ANALYSIS, THE JOURNAL OF SYMBOLIC LOGIC 242–56 (September, 1953), *available at* http://www.chomsky.info/articles/195309—.pdf, and THREE MODELS FOR THE DESCRIPTION OF LANGUAGE, IRE TRANSACTIONS ON INFORMATION THEORY 113–24 (September, 1956).

[331] As the patent application might describe it.

[332] And also gestural languages like American Sign Language.

to any one of those devices, if novel and not obvious can be patented, but that does not mean, despite what the Federal Circuit said in *State Street*, that instructions on how to use those devices—that the processes of using those devices—can be patented.

For our immediate purpose of understanding the workings of modern digital computers, and their programs, the most important characters or glyphs are those that we use to represent numbers. Digital computers, after all, at a fundamental level, deal only with digits, that is, with numerals.'[333] The numerals that we are most familiar with are the so-called "Arabic[334] numerals."

A key feature of the Arabic numerals as we know them is that they contain the glyph "0" standing for the number "zero" that represents the number of elements in a set that does not contain any elements[335] The history of the invention[336] of the number zero is mysterious,[337] but as a revealing thought experiment, one might wonder whether a similar invention would be patentable today under the precedents of *Alappat* and *State Street* and, if it were, how it would effect our ability to think about numbers or do arithmetic.

The use of zero to represent a number is not, however, its primary use:

---

[333] Those numerals may, of course, be an encoded representation of forms of information other than numbers. Thus this article, for example, before it was printed out on paper or on a computer screen, was stored as a sequence of digits on a hard disk of a computer.

[334] The way things are going geo-politically I fear that it may become politically correct to call those numerals "democratic numerals" just as "sauerkraut" became "liberty cabbage" during World War I. In any case, for historical accuracy, we should probably refer to them as "Hindu numerals." Arabic numerals,, known formally as Hindu-Arabic numerals, and also known as Indian numerals, Hindu numerals, European numerals, and Western numerals, are the most common symbolic representation of numbers around the world. They are considered an important milestone in the development of mathematics. One may distinguish between the decimal system involved, also known as the Hindu-Arabic numeral system, and the precise glyphs used. The glyphs most commonly used in conjunction with the Latin alphabet since Early Modern times are 0 1 2 3 4 5 6 7 8 9. The numbers were developed in India by the Hindus around 400 BCE. However, because it was Arabs who relayed this system to the West after the Hindu numerical system found its way to Baghdad, the numeral system became misidentified as "Arabic" in the eyes of the Europeans. . . . Wikipedia, *Arabic Numerals*, http://en.wikipedia.org/wiki/Arabic numerals.

[335] Which mathematicians call the "null set."

[336] Or was it a discovery?

[337] One of the commonest questions which the readers of this archive ask is: Who discovered zero? Why then have we not written an article on zero as one of the first in the archive? The reason is basically because of the difficulty of answering the question in a satisfactory form. If someone had come up with the concept of zero which everyone then saw as a brilliant innovation to enter mathematics from that time on, the question would have a satisfactory answer even if we did not know which genius invented it. The historical record, however, shows quite a different path towards the concept. Zero makes shadowy appearances only to vanish again almost as if mathematicians were searching for it yet did not recognize its fundamental significance even when they saw it. *A History of Zero*, http://tinyurl.com/5tgqy.

> The first thing to say about zero is that there are two uses of zero which are both extremely important but are somewhat different. One use is as an empty place indicator in our place-value number system. Hence in a number like 2106 the zero is used so that the positions of the 2 and 1 are correct. Clearly 216 means something quite different. The second use of zero is as a number itself in the form we use it as 0.[338]

Those of us who are not mathematicians could get along fairly well without having a numeral to represent the number zero, but we would be in terrible shape without the place holder that allows us to distinguish between 216 and 2106. Just think what it would be like to fill in your income tax return using the Roman numeral system, which lacks any means of representing the digit zero in a positional notation system.[339]

Today we still use the decimal system of "Arabic" numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 when we are doing arithmetic by "head and hand"; our computers, on the other hand, use a binary system consisting of only two numbers that can be represented by the numerals 0 and 1. The numbering system using binary digits is known, not surprisingly, as the "binary system." Normally, when one wants to represent those digits on a piece of paper or a blackboard, it is simplest to write them using the familiar figures "0" and "1", but is also possible to treat any pair of signs as a pair of binary digits; thus the pairs "o" and "x", "True" and False", and "Off" and "On" can also be viewed as pairs of binary digits or "bits".[340] A binary number can be also represented by the presence or absence of a hole on a punch card or a paper tape, or by the presence of absence of a magnetized spot on a floppy disk, or by any other pair of discrete signs, including differing electrical potentials.[341]

---

[338] *Id.*

[339] *See* Wikipedia, *Roman Numerals*, http://tinyurl.com/oj9zg.

[340] The term "bit" is a contraction of the phrase "binary digit.

[341] Donald E. Knuth gives a short history of the use of binary notation, going back to times before anyone dreamed of anything like the modern digital computer. The binary system of notation has its own interesting history. Many primitive tribes in existence today are known to use a binary or "pair" system of counting (making groups of two instead of five or ten), but they do not count in a true radix-2 system, since they do not treat powers of two in a special manner.

> The first known appearance of pure binary notation was about 1605 in some unpublished manuscripts of Thomas Harriot (1560–1621). Harriot was a creative man, who first became famous by coming to America as a representative of Sir Walter Raleigh. He invented (among other things) a notation like that now used for "less than" and "greater than" relations; but for some reason he chose not to publish many of his discoveries. . . . The first published discussion of the binary system was given in a comparatively little-known work by a Spanish Bishop, Juan Caramuel Lobkowitz, Mathesis biceps 1 (Campaniæ, 1670), 45–48; Caramuel discussed the

Although a few early digital computers like the ENIAC, and some IBM mainframe computers, do arithmetic using the base 10 decimal system, almost all modern ones do arithmetic and other calculations using streams and strings of binary digits. That means that it is often necessary for us to convert numbers from the decimal system that we use to the binary system that our computers use. And that, of course, explains why IBM wanted to get a patent on the means of converting from binary coded decimal numbers to pure binary representations that was at issue in *Benson*.[342]

And the fact that we can do that conversion in our heads with the aid of pencil and paper is one of the reasons that the Supreme Court gave in *Benson* for refusing to allow a patent on that process.

(ii) *Control Systems*

There are programmable devices that are not used to make computations, but rather are used to control various other mechanical processes. Being programmable they potentially at least raise some of the same legal issues as do computer programs. Historically the most important of these was the Jacquard Loom, which was the ancestor of the "punched card" tabulating machines developed by Hollerith and

---

representation of numbers in radices 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, and 60 at some length, but gave no examples of arithmetic operations in nondecimal systems (except for the trivial operation of adding unity). Ultimately, an article by G.W. Leibnitz [Memoires de l'Academie Royale des Sciences (Paris: 1703), 110 116], which illustrated binary addition, subtraction, multiplication, and division, really brought binary notation into the limelight, and this article is usually referred to as the birth of radix-2 arithmetic. Leibnitz later referred to the binary system frequently. He did not recommend it for practical calculations, but he stressed its importance in number-theoretical investigations, since patterns in number sequences are often more apparent in binary notation than they are in decimal; he also saw a mystical significance in the fact that everything is expressible in terms of zero and one. . . . A careful study of Leibnitz's early work with binary numbers has been made by Hans J. Zacher . . . . Zacher points out that Leibnitz was familiar with John Napier's so-called "local arithmetic," a way for calculating with stones that amounts to using a radix-2 abacus. [Napier had published the idea of local arithmetic in an appendix to his little book Rhabdologia in 1617; it may be called the world's first "binary computer," and it is surely the world's cheapest, although Napier felt that it was more amusing than practical. . . .]

. . . .

Charles XII of Sweden, whose talent for mathematics exceeded that of all other kings in the history of the world, hit on the idea of radix-8 arithmetic about 1717. This was probably his own invention, although he had met Leibnitz briefly in 1707. Charles felt that radix 8 or 64 would be more convenient for calculation than the decimal system, and he considered introducing octal arithmetic into Sweden; but he died in battle before decreeing such a change. . . .

[342] *See supra* Part II.A.

forever associated with IBM.[343] And the idea of using punched cards like those used to control the Jacquard Loom was also adopted by Babbage in his Analytical Engine.[344]

The Jacquard Loom was a machine, like other looms, that was used for weaving fabrics. The Jacquard loom, unlike any earlier loom, could be programmed to weave a particular pattern without additional human intervention once the program was written[345] and inserted into a mechanism that controlled the operations of the loom. It was the first machine to be controlled by a "program" that was punched into punch cards, an input device that were later adapted by Babbage for use in his "analytical engine"[346] and by Hollerith when he designed his tabulation machines.[347]

---

[343] *See supra* Part VI.A.8.

[344] *See supra* Part VI.A.3(a)(ii).

[345] That is, punched as a pattern of holes on a punch card.

[346] *See supra* Part VI.A.3(a)(i).

[347] *See supra* note 291. Here is a description of the workings of Jacquard loom written in 1842 as part of an essay on the workings of Babbage's analytical engine:

> Two species of threads are usually distinguished in woven stuffs; one is the warp or longitudinal thread, the other the woof or transverse thread, which is conveyed by the instrument called the shuttle, and which crosses the longitudinal thread or warp. When a brocaded stuff is required, it is necessary in turn to prevent certain threads from crossing the woof, and this according to a succession which is determined by the nature of the design that is to be reproduced. Formerly this process was lengthy and difficult, and it was requisite that the workman, by attending to the design which he was to copy, should himself regulate the movements the threads were to take. Thence arose the high price of this description of stuffs, especially if threads of various colours entered into the fabric. To simplify this manufacture, Jacquard devised the plan of connecting each group of threads that were to act together, with a distinct lever belonging exclusively to that group. All these levers terminate in rods, which are united together in one bundle, having usually the form of a parallelopiped with a rectangular base. The rods are cylindrical, and are separated from each other by small intervals. The process of raising the threads is thus resolved into that of moving these various lever-arms in the requisite order. To effect this, a rectangular sheet of pasteboard is taken, somewhat larger in size than a Part of the bundle of lever-arms. If this sheet be applied to the base of the bundle, and an advancing motion be then communicated to the pasteboard, this latter will move with it all the rods of the bundle, and consequently the threads that are connected with each of them. But if the pasteboard, instead of being plain, were pierced with holes corresponding to the extremities of the levers which meet it, then, since each of the levers would pass through the pasteboard during the motion of the latter, they would all remain in their places. We thus see that it is easy so to determine the position of the holes in the pasteboard, that, at any given moment, there shall be a certain number of levers, and consequently of parcels of threads, raised, while the rest remain where they were. Supposing this process is successively repeated according to a law indicated by the pattern to be executed, we perceive that this pattern may be reproduced on the stuff. For this purpose we need merely compose a series of cards according to the law required, and arrange them in suitable order one after the other; then, by causing them to pass over a polygonal beam which is so connected as to turn a new face for every stroke of the shuttle, which face shall then be impelled parallelly to itself against the bundle of lever-arms, the operation of raising the

Does *State Street* hold that the instructions fed into a Jacquard loom to produce a particular pattern can be patented?[348]

If not, why not?

The key point here is not that the loom was programmable but that the only thing that was novel about its output was a pattern or a design, not the cloth that was just like the material produced by other non-programmable looms. Such a pattern clearly should be no more patentable than the music stored or a CD-ROM or created by a Moog synthesizer.[349]

More modern examples of devices like the Jacquard loom are programmable machine tools, that originally were controlled by programs stored on punch cards but that today may be directly connected to computers. The only significant difference between machine tools and the Jacquard loom is that the former turn out parts used in traditional machines, rather than cloth with patterns woven into it. Thus the material products of the machine tools, and even the mechanical processes that create those products, may be patentable, just as they would be patentable if they were implemented by a human machinist rather than by a machine. Computer programs are not unpatentable because they are instructions to a machine; computer programs are unpatentable because they are instructions to a machine to cause it to process information.

4. Modern Digital Computers

In the United States the first Turing complete electronic digital computer[350] was the ENIAC, the Electronic Numerical Integrator and Computer.

> threads will be regularly performed. Thus we see that brocaded tissues may be manufactured with a precision and rapidity formerly difficult to obtain.

L.F. Menabrea, *Sketch of The Analytical Engine Invented by Charles Babbage*, http://www.fourmilab.ch/babbage/sketch.html.

[348] We are not concerned here with so-called "design patents" that might—but probably would not—allow the pattern itself to be patented.

[349] See Chief Judge Archer's remarks quoted in *supra* note 121.

[350] The first such computer was constructed by Konrad Zuse, who completed the Z3 in 1941, with recycled materials donated by fellow university staff and students. This was the world's first electronic, fully programmable digital computer based on a binary floating-point number and switching system. Zuse used old movie film to store his programs and data for the Z3, instead of using paper tape or punched cards. Paper was in short supply in Germany during the war. Mary Belis, *Inventors of the Modern Computer*, http://inventors.about.com/library/weekly/aa050298.htm. Konrad Zuse [also] designed the high-level programming language Plankalkül (Calculus of Programs) in 1945, after moving out of Berlin at the end of World War II. Anyone who has had the opportunity to study the original definition of Plankalkül is struck

> . . . . When it was finished, the ENIAC filled an entire room, weighed thirty tons, and consumed two hundred kilowatts of power. It generated so much heat that it had to be placed in one of the few rooms at the University with a forced air cooling system. Vacuum tubes, over 19,000 of them, were the principal elements in the computer's circuitry. It also had fifteen hundred relays and hundreds of thousands of resistors, capacitors, and inductors. All of this electronics were held in forty-two panels nine feet tall, two feet wide, and one foot thick. They were arranged in a "U" shape, with three panels on wheels so they could be moved around. An IBM card reader and card punch were used respectively for input and output.
>
> . . . .
>
> The ENIAC was programmed by wiring cable connections and setting three thousand switches on the function tables. This had to be done for every problem and made using the machine very tedious. However, the speed of the computation made up for this. Ballistic trajectories can take someone with a hand calculator twenty hours to compute. . . . The ENIAC could do it in thirty seconds.[351]
>
> For all their apparent complexity, digital computers are basically simple machines. Every operation they perform, from navigating a spacecraft to playing a game of chess, is based on one key operation: determining whether certain electronic switches, called gates, are open or closed. The real power of a computer lies in the speed with which it checks these switches.
>
> A computer can recognize only two states in each of its millions of circuit switches–on or off, or high voltage or low voltage. By assigning binary numbers to these states–1 for on

---

by its modern flavor and powerful constructs—it seems as if it had been created much later than 1945. Most amazing, however, is the fact that at the time that Konrad Zuse was writing his Plankalkül document, the only two working computers in the world were the ENIAC and the Harvard Mark I. None of them used a compiler or a formula translator—the ENIAC had even to be rewired for every different problem. Raúl Rojas et al., Plankalkül: *The First High-Level Programming Language and its Implementation*, http://tinyurl.com/nregf.

[351] Kevin W. Richey, *The ENIAC*, http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML. Apparently it was the fact that the ENIAC could only be reprogrammed by rewiring it that led to the perverse idea that programs are machines.

> and 0 for off, for example–and linking many switches together, a computer can represent any type of data, from numbers to letters to musical notes. This process is called digitization.[352]

Exactly how the "hardware" that makes up the tangible computer works or out of what it is made are matters beyond the scope of this article. In my lifetime the switches used in digital computers have taken the form of vacuum tubes—or valves, as the British call them—wired together in circuits with other components like resistors and capacitors, and then the valves were replaced by individual transistors, which were much smaller, consumed less energy, and were longer lasting, and then the transistors and other components, including the wiring that connected them, were embodied in the now familiar "microchip" etched onto a substrate of silicon. These basic components that are used in computers continue to be refined and may in time be replaced by entirely new technologies, but, though the material structure of the switches may change, the logical functions of those switches are likely to remain constant.[353]

---

[352] *computer*, BRITANNICA STUDENT ENCYCLOPEDIA. (2005), *available at* http://search.eb.com/ebi/article?tocId=199023.

[353] If the current electrical switches are replaced by optical switches as will probably happen in some cases, the logical functioning of the switches will not change and no computer programs will need to be rewritten except those that function at the lowest level of drivers that direct the flow of information to, and fetch information from, the hardware devices. The potential development of "quantum computers" or other highly parallel devices may require different algorithms for conducting certain mathematical operations, and thus may require new programs, but such a development will not change the fundamental nature of computer programs or programming. Behold your computer. Your computer represents the culmination of years of technological advancements beginning with the early ideas of Charles Babbage (1791–1871) and eventual creation of the first computer by German engineer Konrad Zuse in 1941. Surprisingly however, the high speed modern computer sitting in front of you is fundamentally no different from its gargantuan 30 ton ancestors, which were equipped with some 18000 vacuum tubes and 500 miles of wiring! Although computers have become more compact and considerably faster in performing their task, the task remains the same: to manipulate and interpret an encoding of binary bits into a useful computational result. A bit is a fundamental unit of information, classically represented as a 0 or 1 in your digital computer. Each classical bit is physically realized through a macroscopic physical system, such as the magnetization on a hard disk or the charge on a capacitor. A document, for example, comprised of n-characters stored on the hard drive of a typical computer is accordingly described by a string of 8n zeros and ones. Herein lies a key difference between your classical computer and a quantum computer. Where a classical computer obeys the well understood laws of classical physics, a quantum computer is a device that harnesses physical phenomenon unique to quantum mechanics (especially quantum interference) to realize a fundamentally new mode of information processing. In a quantum computer, the fundamental unit of information (called a quantum bit or qubit), is not binary but rather more quaternary in nature. This qubit property arises as a direct consequence of its adherence to the laws of quantum mechanics which differ radically from the laws of classical physics. A qubit can exist not only in a state corresponding to the logical state 0 or 1 as in a classical bit, but also in states corresponding to a blend or superposition of these classical states. In other words, a qubit can exist as a zero, a one, or

How these switches or logical gates function is discussed briefly in Section VI.A.5(b).

5. Boolean Logic

Let us cease discussing hardware, for a while at least, and turn our attention to what it is that we do with computers, whatever their hardware may be made of.

When one thinks of what we do with computers at a fundamental level one is likely to conclude that they are used to perform computations, that is, that they are used to do arithmetic. A more accurate description, however, is that we use them to perform logical operations which are mathematical, if not necessarily arithmetical, in nature.

> Around the same time that Charles Babbage was struggling with his Analytical Engine,[354] one of his contemporaries, a British mathematician called George Boole, was busily inventing a new and rather cunning form of mathematics. Boole made significant contributions in several areas of mathematics, but was immortalized for two works in 1847 and 1854, in which he represented logical expressions in a mathematical form now known as Boolean Algebra. Boole's

---

simultaneously as both 0 and 1, with a numerical coefficient representing the probability for each state. This may seem counterintuitive because everyday phenomenon are governed by classical physics, not quantum mechanics – which takes over at the atomic level. . . .

. . . .

> In a traditional computer, information is encoded in a series of bits, and these bits are manipulated via Boolean logic gates arranged in succession to produce an end result. Similarly, a quantum computer manipulates qubits by executing a series of quantum gates, each a unitary transformation acting on a single qubit or pair of qubits. In applying these gates in succession, a quantum computer can perform a complicated unitary transformation to a set of qubits in some initial state. The qubits can then be measured, with this measurement serving as the final computational result. This similarity in calculation between a classical and quantum computer affords that in theory, a classical computer can accurately simulate a quantum computer. In other words, a classical computer would be able to do anything a quantum computer can. So why bother with quantum computers? Although a classical computer can theoretically simulate a quantum computer, it is incredibly inefficient, so much so that a classical computer is effectively incapable of performing many tasks that a quantum computer could perform with ease. The simulation of a quantum computer on a classical one is a computationally hard problem because the correlations among quantum bits are qualitatively different from correlations among classical bits . . . ,

Jacob West, *The Quantum Computer*, http://tinyurl.com/37b8.

[354] *See supra* Part VI.1.C(a)(i).

> work was all the more impressive because, with the exception of elementary school and a short time in a commercial school, he was almost completely self-educated.
>
> In conjunction with Boole, another British mathematician, Augustus DeMorgan, formalized a set of logical operations now known as DeMorgan transformations. As the Encyclopedia Britannica says: "A renascence of logical studies came about almost entirely because of Boole and DeMorgan."
>
> . . . Unfortunately, with the exception of students of philosophy and symbolic logic, Boolean Algebra was destined to remain largely unknown and unused for the better part of a century, until a young student called Claude E. Shannon recognized its relevance to electronics design.[355]

A key development in the history of the modern digital computer was the recognition by Claude Shannon[356] in his 1938 Masters thesis at MIT[357] of the fact that the behavior of switching circuits of the sort that later were to become the key components in electronic computers—which did not exist at that time—could be described using Boolean algebra,[358] Modern digital computers typically are programmed to perform Boolean, rather than arithmetical, operations and often have separate processing units to deal with arithmetical operations that are not really part of the Central Processing Unit.

Boolean algebra deals with certain logical operators like AND and OR and NOT that we use every day in an unformalized fashion. If A is true and if B is true then we know, instinctively as it were, that A AND B is true. On the other hand, we also know that if A is false then, whether or not B is true or false, it is not true—*i.e.*, it is false—that A AND B is true.[359]

For those of us who are not concerned with low level programming and thus do not have to worry about the functioning of the logic gates in a computer, our experience with Boolean logic is

---

[355] These notes are abstracted from the book BEBOP BYTES BACK (AN UNCONVENTIONAL GUIDE TO COMPUTERS), *available at* http://www.maxmon.com/1847ad.htm.

[356] Shannon was also the author of the seminal work on communication theory that first explored the modern concept of "information" as that term is used in computer science.

[357] Which has been described as the most important master's thesis ever written.

[358] C. Shannon, *A symbolic analysis of relay and switching circuits*, https://dspace.mit.edu/handle/1721.1/11173.

[359] The binary values "true" and "false" can equally well be represented by the binary digits—or "bits"—"0" and "1". note that these two binary values correspond to the two symbols that are used by Turing machines. *See supra* note 315.

likely to involve searching for particular items of computerized information. Lawyers and law students, though they may not realize it, use Boolean logic every time they make a search through the legal materials stored in the Lexis or Westlaw databases and anyone who uses the advanced search function on Google is also making a Boolean search: "Find results with all of the words" causes Google to return all texts in which it is true that word1 AND word2 AND word3, etc., appear while "Find results with at least one of the words" causes Google to return all texts in which it is true that word1 OR word2 OR word3 appear.

All relational databases, such as PostgreSQL or Microsoft SQL,[360] and almost all other types of databases, also permit Boolean searching.

Boolean operations are best understood in terms of so-called "Truth Tables," the subject discussed in the next section.

a. Truth Tables

Around 1921 Emil Post[361] and Ludwig Wittgenstein[362] independently developed the idea of using "Truth Tables" to determine whether a compound statement such as "swans are birds AND swans are black" is true or false.[363]

The simplest sort of truth table reveals whether a simple statement like "swans are birds" is true. There are two possible operations that can apply to such a simple statement, the IDENTITY or IS operator and the NOT operator. Here is the truth table for the IS operator, which, since all it shows is that A is true if A is true and that A is false if A is false, is not very useful:

A	IS A
T	T
F	F

On the other hand, the truth table for the NOT operator, which shows that NOT A is false if A is true—and vice versa—is, as we shall see, very useful when used in conjunction with other Boolean operators:

---

[360] "SQL" stands for "Standard Query Language."

[361] *See* http://www-groups.dcs.st-and.ac.uk/ history/Mathematicians/Post.html.

[362] *See* http://tinyurl.com/m3ess.

[363] Since, although it is true that swans are birds, it is not true that swans are black this compound statement is false.

A	NOT A
T	F
F	T

Here are the truth tables for the other Boolean operators with which we are most familiar, OR and AND:

A	B	A OR B
T	T	T
T	F	T
F	T	T
F	F	F

A	B	A AND B
T	T	T
T	F	F
F	T	F
F	F	F

As you can see, A OR B is true in all cases except when both A and B are false, while A AND B is false in all cases except when both A and B are true. At this point you may be wondering why we have wandered off from the subject of computers and instead are examining the truth tables for various Boolean operators. The reason, however, is quite simple: All the instructions carried out or operations performed by modern digital computers or a Turing machine—even arithmetical operations—can be specified as a combination of Boolean operations.[364]

Truth tables can be used not only to specify truth or falsity: they can be applied to any disjoint pair of relations like "yes or no," "on or off," "up or down," or, in the case of binary digits, "1 or 0." Since modern digital computers process their inputs in the form of binary digits and also produce their outputs in that form, I shall hereafter discuss truth tables using "1" instead of "T" and "0" instead of "F."[365]

One important fact about truth tables—important at least when one is designing the wiring of a digital computer—is that every Boolean operator can be produced by some combination of the NOT and AND—or the NOT and OR—operators. And there are two additional operators, NOR (NOT OR) and NAND (NOT AND), either one of which can be used to produce all other Boolean operators.

[364] *See supra* note 314 and accompanying text and *infra* Part VI.A.5(b).

[365] It does, after all, not matter what symbols we use to designate the two different voltage levels that can be the input to, or the output of, a digital logic gate, for those symbols, no matter how we interpret them, are not voltage levels.

Here are the truth tables for NOR and NAND:[366]

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

A	B	A AND B
1	1	0
1	0	1
0	1	1
0	0	1

There are two additional Boolean operator besides NOT, OR, AND, NOR, and AND that we should discuss: the operation referred as XOR and the one referred to as IF/THEN.

The XOR operator performs the so-called "exclusive or" operation—also known as "exclusive disjunction"—where the result is true whenever either A or B, but not the other, is true. Here is the truth table for the XOR operation:[367]

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

One important use of the XOR operation is in cryptography, the science or art of encoding messages or other information so that they cannot be understood without the application of some additional secret information. One of the simplest ways of encrypting a message is to use a so-called "one-time pad" where each bit of information in the message is XORed[368] together with the corresponding bit in another file called the "key."[369]

It was my writing a little program to demonstrate this fact to my students in my course in computing and the law that lead to my suit against the United States Secretary of Commerce to establish the fact that the First Amendment to the constitution of the United States protects the publication of information in the form of computer programs, even on the Internet, from governmental censorship.[370] For

[366] Note that "A NAND A" has the same truth table as "NOT A" and that "NOT(A NAND B)" has the same truth table as "A AND B." The truth tables for all Boolean operators can be created in a similar fashion.

[367] Remember that we are now using the symbols "1" for "true" and "0" for "false."

[368] Or some equivalent operation is used like NOT XOR.

[369] If the key is truly random and never reused, this is the one form of encryption that cannot be broken, even in theory, unless, of course, some enemy agent steals a copy of the key.

[370] Junger v. Daley, 209 F. 3d 481 (6th Cir. 2000).

our immediate purposes, one might wonder if the First Amendment would protect me if it were a holder of a patent on such encryption programs who tried to enjoin my publication of the text of my program?

One of the most important Boolean operations when performed by a digital computer is the IF AND ONLY IF operation, where the result is true if and only if both A and B are true.

Here is the truth table for the IF AND ONLY IF operator:

A	B	A IF AND ONLY IF B
1	1	1
1	0	0
0	1	0
0	0	0

> This operation allows one to instruct a computer, human or electronic, to do something if and only if A and B are both true.[371]

Now this discussion has probably told you more than you, or I, will ever need to know about Boolean operators and truth tables. But it does serve to make a critical point: It is ridiculous to think that a patent could be issued for a string of Boolean operations implemented by looking up their outcomes in a set of truth tables. Yet, in theory at least, every program for a binary computer can be implemented in exactly that fashion.

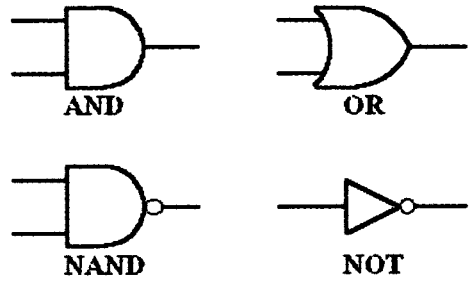
b. Wiring Diagrams and Logic Gates

Where logicians use truth tables, electrical engineers, when designing computers and related devices, use wiring diagrams consisting of a representation of the wires in the device connected with each other by so-called "logic gates," which typically take two inputs which are either "high voltage" or "low voltage" and have a single output, again either high or low. "High" traditionally corresponds to "1" or "True" while "low" corresponds to "0" or "False" in a truth table. Here are some examples of the way logic gates are represented:[372]

---

[371] Or to do something where A = B and A is true.

[372] David Warner Smith, *Digital Logic Systems*, http://users.senet.com.au/~dwsmith/beginners.htm.



A computer program, particularly one that is hard-wired into a device, can thus be represented by a wiring diagram, so, though it is ridiculous to claim that a computer program is a device like the wiring in a telephone, it is quite reasonable, and sometimes even useful, to claim that a computer program is a schematic like a wiring diagram,[373] provided always that you keep in mind that the output of the computer running the program is going to consist of nothing but a bunch of ones and zeros[374]

c. Flowcharts

Closely related to wiring diagrams are flowcharts, which represent, not the flow of signals through the computer, but rather the logical steps that the computer will take.[375] I gather that flowcharts are not

---

[373] For more information about logic gates and their relation to truth tables, you might want to look at Douglas W. Jones, *Computer Organization Collection*, http://www.cs.uiowa.edu/~jones/assem/notes/08arith.shtml.

[374] Ones and zeros, though they may be represented by holes in a punch card or voltage levels in a computer chip, do not themselves consist of matter or energy. They are pure information.

[375] A typical flowchart from older Computer Science textbooks may have the following kinds of symbols:

- Start and end symbols, represented as lozenges, ovals or rounded rectangles, usually containing the word "Start" or "End".

- Arrows, showing what's called "flow of control" in computer science. An arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to.

- Processing steps, represented as rectangles. Example: Add 1 to X.

- Input/Output, represented as a parallelogram. Examples: Get X from the user; display X.

- Conditional (or decision), represented as a diamond (rhombus). These typically contain a Yes/No question or True/False test. This symbol is unique in that it has two arrows coming out of it, usually from the bottom point and right point, one corresponding to Yes or True, and one corresponding to No or False. The arrows

used so often these days, but back when I took the IBM Executive Computer Concepts Course[376] we were required to draw a flowchart showing what our computer would do before writing down the actual program.[377]

Flowcharts have the advantage that, though they can easily be transformed into executable code, they are not limited to the architecture of any particular machine or the idiosyncrasies of any particular programming language.[378] Since a flowchart is neither a machine nor a process, it is difficult to imagine how even the Federal Circuit would deem it to be patentable or how drawing such a chart should amount to "making" a patented computer program and thus infringing a patent.

Yet every computer program can be represented by a flow chart.[379]

6. Registers

Computer programs operate on binary digits or "bits"[380] and most operations that computer hardware can perform—and thus that computer programs can usefully specify—involve a series of ordered bits: eight bits (a "byte"), sixteen bits, thirty-two bits, or sixty-four bits[381] that are manipulated as a single unit in "registers" that typically are part of the computer's central processing unit. One of the important things that I learned in the IBM Executive Computer Concepts course[382] that I took back in the 1960's was that when

---

> should always be labeled. Flowcharts may contain other symbols, such as connectors, usually represented as circles, to represent converging paths in the flow chart. Circles will have more than one arrow coming into them but only one going out. Some flow charts may just have an arrow point to another arrow instead. These are useful to represent an iterative process (what in Computer Science is called a loop). A loop may, for example, consist of a connector where control first enters, processing steps, a conditional with one arrow exiting the loop, and one going back to the connector.

Wikipedia, *Flowchart*, http://en.wikipedia.org/wiki/Flowchart.

[376] *See infra* note 387.

[377] And we had to write our program on coding sheets before encoding it in punch cards using a cardpunch machine.

[378] The same result can be obtained by using psuedocode, "Pseudocode (derived from pseudo and code) is a description of a computer programming algorithm that uses the structural conventions of programming languages, but omits detailed subroutines or language-specific syntax." Wikipedia. *Psuedocode*, http://en.wikipedia.org/wiki/Psuedocode.

[379] Or by a description of the program in psuedocode.

[380] *See supra* note 343 and accompanying text.

[381] Note that each of these lengths is a power of 2: $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, and $64 = 2^6$.

[382] That course only lasted a week, but it was by far the most exhausting course that I have ever taken. It was not designed for relatively junior lawyers like myself, but for what are now called Chief Financial Officers, who apparently cannot stand not being overworked. The course

writing a program at the fundamental level of machine code or assembly language, all instructions to an electronic computer must specify the register or registers that are to be used in implementing each instruction.

For example, a computer with the appropriate processor will understand this x86/IA-32 machine language:

> 10110000 01100001
>
> For programmers, however, it is easier to remember the equivalent assembly language representation:
>
> mov al, 061h
>
> which means to move the hexadecimal value 61 (97 decimal) into the processor register with the name "al". The mnemonic "mov" is short for "move", and a comma-separated list of arguments or parameters follows it; this is a typical assembly language statement.[383]

When writing code in "higher level" languages,[384] the programmer does not have to worry about moving data in and out of registers or know which register contains which chunk of data, for these matters are taken care of automatically by the compiler or interpreter that translate the instructions in the higher level language into code—"machine code[385]"—that the hardware can implement.

Some operations may be limited to a specific register—for example, the Intel 8080[386] chip could perform addition only in a

---

took the form of a series of time capsules: the first day we had to program in machine language, where all instructions took the form of a string of numbers that were fed into the computer's registers, the next day they invented assembly language, which allowed one to substitute strings of alphanumeric characters, that sort of look like words, for the strings of numbers in the corresponding machine language version of the program, *see* Wikipedia, *Assembly language*, the next day they invented Fortran, *see* Wikipedia, *FORTRAN*, the next day they invented COBOL, *see* Wikipedia, *COBOL*, and so on. After writing our programs we had to punch them into a deck of punch cards that could be fed into the computer. On the last day they were going to show us a terminal with a CRT screen and a keyboard, but unfortunately on that day IBM's terminal in New York City was broken.

[383] Wikipedia, *Assembly language*, http://en.wikipedia.org/wiki/Assembly_language.

[384] That is, languages other than assembly language or machine language.

[385] Or, sometimes, into assembly language that an assembler then converts into machine code.

[386] The first computer that I owned was a NorthStar Horizon which had a Zilog Z80 chip that had the same instruction set as the Intel 8080, so here I am writing from personal experience. The assembler that I used was the CP/M ASM assembler; a copy of its manual can be found at http://www.autometer.de/gaby/DRI/ASM.pdf.

specialized register called the "accumulator,"[387] A register that is of particular interest to us is, of course, a "shift register," since one of the claims that was held in *Gottschalk v. Benson*[388] to be unpatentable involved the use of a "reentrant shift register,"[389] a register[390] in which a string of binary digits could be shifted to the left or to the right. Shifting the string one place to the left amounts to multiplying the number represented by the string by two; shifting it one place to the right amounts to dividing it by two. Thus, for example, if a register contains the number five in binary notation, its contents would look like this:

00000101

after shifting the contents one place to the left, they would look like this:

00001010

which equals the number ten.

The claim in *Benson*, which did not relate to the design or construction of any type of physical register, thus amounted to little more than a series of multiplications by two and divisions by two of a number that originally represented a Binary Coded Decimal version of another number. It is hardly surprising that the Supreme Court held that such a series of arithmetical operations is not the sort of process that can be patented.

When one gets right down to the hardware, all the code—all of the instructions—and all the data that are fed into a computer take the form of strings of binary digits—bits—that are processed in some way in the registers of a computer's central processing unit and then output as another string of bits.

7. Peripherals

Registers are at the very heart of the computer; on the other hand there are peripherals like video screens and keyboards and printers that, though often connected to a computer, are not parts of it. These peripherals are of critical importance to the use that we make of

---

[387] So called because it could store and accumulate intermediate results.

[388] *See supra* Part III.

[389] *See supra* note 43 and accompanying text.

[390] In the case of the Intel 8080 that register was the accumulator. The instruction to shift the contents of the accumulator to the left was "RAL" and to the right was RAR, standing for "Rotate Accumulator Left" and "Rotate Accumulator Right," respectively.

computers, because they let us feed information into the computer and, once it has been processed, get it out again, but, once again, they are not themselves parts of the computer.[391]

There is a real danger, however, that a person who has only dealt with a computer using a pointing device like a mouse and a video screen with a Graphical User Interface—a "GUI"[392]—will confuse the peripheral devices with the computer itself.

The relation between the computer and its peripherals is perhaps best illustrated by the examples of the keyboard and an early printer such as the one that I had connected to my first computer, back in the days before the introduction of the IBM PC.[393] When one depressed a key on the keyboard, say the key marked A, that device sent a code—a number in the form of a string of binary digits—to the computer and, if the computer was running a program that edits text, like the WordStar program that I used as a word processor, that program would insert the ASCII code for the letter[394] in the file that it was

---

[391] Although, of course, they may like modern washing machines and toasters contain a computer themselves.

[392] Also known as a "WIMP" environment. WIMP environment n. [acronym: 'Window, Icon, Menu, Pointing device (or Pull-down menu)'] A graphical-user-interface environment such as X or the Macintosh interface, esp. as described by a hacker who prefers command-line interfaces for their superior flexibility and extensibility. However, it is also used without negative connotations; one must pay attention to voice tone and other signals to interpret correctly.

[393] *See supra* note 389.

[394] Or, if the shift key were depressed, the ASCII code for the letter A. ASCII : /askee/, n. [originally an acronym (American Standard Code for Information Interchange) but now merely conventional] The predominant character set encoding of present-day computers. The standard version uses 7 bits for each character, whereas most earlier codes (including early drafts of ASCII prior to June 1961) used fewer. This change allowed the inclusion of lowercase letters . . . but it did not provide for accented letters or any other letterforms not used in English (such as the German sharp-S ß or the ae-ligature æ which is a letter in, for example, Norwegian). It could be worse, though. It could be much worse. For a listing of the various ASCII codes, see *ASCII Table and Description*, http://www.lookuptables.com/. This table begins with the following comment:

> ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as tabs, bold or underscoring—the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. . . .

For a history of ASCII and its predecessors, including Morse Code, see Tom Jennings, *ASCII: American Standard Code for Information Infiltration*, http://www.wps.com/projects/codes/.

editing. And then, later, when the file was sent to the printer and the printer got to that segment of the code it would print the letter a.

Thus the keyboard sent a code to the computer, the computer processed that code, and the computer then sent another code to the printer and the printer printed the appropriate character. We need peripheral devices to get information into the computer and to get information out of the computer in a form where we can understand it, but it is important to realize that the peripheral devices are not actually parts of the computer.[395]

8. Near the Beginning Was the Plugboard

Before computers could be instructed to reprogram themselves and before they had central processing units or registers, they could be rewired, as were tabulating machines, using "plugboards."

> The very first punched-card tabulating machines were custom-built to do just one job, such as tally the US 1890 or 1900 census data.[396] Starting in 1906, tabulators were made more flexible by addition of a wiring panel to let users control their actions to some degree, thus allowing the same machine to be sold into different markets (government, railroad, etc) and used for different purposes. But this also meant that if one machine were to be used for different jobs, it would have to be rewired between each job, often a lengthy process that kept the machine offline for extended periods. In 1928, IBM began to manufacture machines with removeable wiring panels, or "plugboards", so programs could be prewired and swapped in and out instantly to change jobs. . . .
>
> By the 1930s, most IBM punched-card equipment—tabulators, accounting machines, multiplying and summary punches, calculators—operated under the control of a plugboard, now formally referred to as a control panel. Users wired plugboards to specify exactly which card columns were to be read or punched, which card fields were to be fed to which accumulators, and so forth, depending on the machine.

---

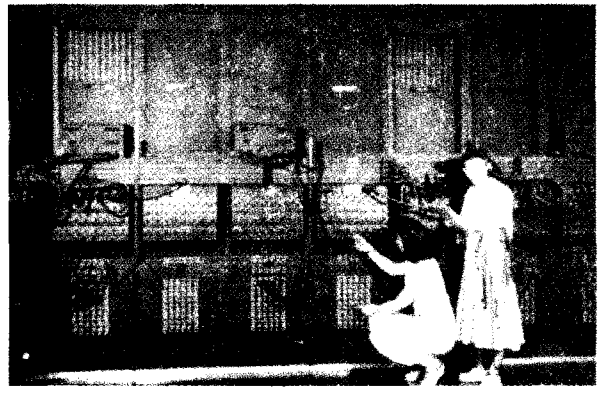
[395] Although nowadays they may contain a computer to assist them in performing their functions. It should also be noted that the Federal Circuit has suggested that information in digital form stored on a "physical structure" like a CD-ROM or player piano role is patentable, not as software but simply as a novel physical structure. *See supra* note 123 and accompanying text.

[396] These tabulating machines were designed by Herman Hollerith, *see* http://www.maxmon.com/1890ad.htm, and were used, as we have seen, by the human computers who worked on the Manhattan project; *see supra* text accompanying note 265.

> The instructions were given by connecting holes ("hubs") with wires. For example, a simple task on an accounting machine might be to print columns of numbers from a deck of cards, in which case a series of wires would connect card columns to printer columns. A slightly more ambitious version of this task would also accumulate totals by connecting the same card columns to accumulators, and then print the totals at the end.[397]

The first electronic computer in the United States is usually considered to have been the ENIAC, which could be reprogrammed by changing the wiring in a plug board. This apparently is what led to the perverse conclusion that the first computer programs were hardware, rather than that they were implemented in hardware (as all computer programs are).[398]

Here is a picture of the ENIAC being programmed by two women:[399]



Hardwired machines like the ENIAC were soon replaced by stored program computers:[400]

---

[397] IBM Control Panels, http://www.columbia.edu/acis/history/plugboard.html.

[398] *See infra* note 415 and accompanying text.

[399] Historic Computer Images, http://ftp.arl.mil/ftp/historic-computers/.

[400] Also known as Von Neumann machines in honor of John von Neumann who first publicized the concept. *See* Arthur W. Burks, Herman H. Goldstine, & John von Neumann, *Preliminary discussion of the logical design of an electronic computing instrument*, http://www.eecs.harvard.edu/jonathan/neumann/neumann.html. Here are excerpts from that publication:

> PART I
>
> 1. Principal components of the machine

1.1. Inasmuch as the completed device will be a general-purpose computing machine it should contain certain main organs relating to arithmetic, memory storage, control and connection with the human operator. It is intended that the machine be fully automatic in character, *i.e.* independent of the human operator after the computation starts. . . .

1.2. It is evident that the machine must be capable of storing in some manner not only the digital information needed in a given computation such as boundary values, tables of functions (such as the equation of state of a fluid) and also the intermediate results of the computation (which may be wanted for varying lengths of time), but also the instructions which govern the actual routine to be performed on the numerical data. In a special-purpose machine these instructions are an integral part of the device and constitute a part of its design structure. For an all-purpose machine it must be possible to instruct the device to carry out any computation that can be formulated in numerical terms. Hence there must be some organ capable of storing these program orders. There must, moreover, be a unit which can understand these instructions and order their execution.

1.3. Conceptually we have discussed above two different forms of memory: storage of numbers and storage of orders. If, however, the orders to the machine are reduced to a numerical code and if the machine can in some fashion distinguish a number from an order, the memory organ can be used to store both numbers and orders. . . .

. . . .

1.5. Inasmuch as the device is to be a computing machine there must be an arithmetic organ in it which can perform certain of the elementary arithmetic operations. There will be, therefore, a unit capable of adding, subtracting, multiplying and dividing. It will be seen . . . that it can also perform additional operations that occur quite frequently. The operations that the machine will view as elementary are clearly those which are wired into the machine. To illustrate, the operation of multiplication could be eliminated from the device as an elementary process if one were willing to view it as a properly ordered series of additions. Similar remarks apply to division. In general, the inner economy of the arithmetic unit is determined by a compromise between the desire for speed of operation—a non-elementary operation will generally take a long time to perform since it is constituted of a series of orders given by the control-and the desire for simplicity, or cheapness, of the machine.

1.6. Lastly there must exist devices, the input and output organ, whereby the human operator and the machine can communicate with each other. . . . 2. First remarks on the memory

. . . .

3. First remarks on the control and code

. . . .

3.3. It must be possible to transfer data from the memory to the arithmetic organ and back again. In transferring information from the arithmetic organ back into the memory there are two types we must distinguish: Transfers of numbers as such and transfers of numbers which are parts of orders. . . .

Almost all modern computers are stored program computers that can be reprogrammed simply by feeding the new program—as a string of binary digits—into the machine's memory. It is this feature that allows one to feed a higher level program—like one written in Fortran—into the computer's memory and then using a compiler

---

. . . .

4. The memory organ

5. The arithmetic organ

5.1. In this Part we discuss the features we now consider desirable for the arithmetic part of our machine. We give our tentative conclusions as to which of the arithmetic operations should be built into the machine and which should be programmed. . . .

5.2. In a discussion of the arithmetical organs of a computing machine one is naturally led to a consideration of the number system to be adopted. In spite of the longstanding tradition of building digital machines in the decimal system, we feel strongly in favor of the binary system for our device. Our fundamental unit of memory is naturally adapted to the binary system since we do not attempt to measure gradations of charge at a particular point . . . but are content to distinguish two states. . . . The main virtue of the binary system as against the decimal is, however, the greater simplicity and speed with which the elementary operations can be performed. To illustrate, consider multiplication by repeated addition. In binary multiplication the product of a particular digit of the multiplier by the multiplicand is either the multiplicand or null according as the multiplier digit is 1 or 0. In the decimal system, however, this product has ten possible values between null and nine times the multiplicand, inclusive. Of course, a decimal umber has only log102 0.3 times as many digits as a binary number of the same accuracy, but even so multiplication in the decimal system is considerably longer than in the binary system. One can accelerate decimal multiplication by complicating the circuits, but this fact is irrelevant to the point just made since binary multiplication can likewise be accelerated by adding to the equipment. Similar remarks may be made about the other operations. In additional point that deserves emphasis is this: An important part of the machine is not arithmetical, but logical in nature. Now logic, being a yes-no system, is fundamentally binary. Therefore a binary arrangement of the arithmetical organs contributes very significantly towards producing a more homogeneous machine, which can be better integrated and is more efficient. The one disadvantage of the binary system from the human point of view is the conversion problem. Since, however, it is completely known how to convert numbers from one base to another and since this conversion can be effected solely by the use of the usual arithmetic processes there is no reason why the computer itself cannot carry out this conversion. . . . Indeed a general-purpose computer, used as a scientific research tool, is called upon to do a very great number of multiplications upon a relatively small amount of input data, and hence the time consumed in the decimal to binary conversion is only a trivial percentage of the total computing time. A similar remark is applicable to the output data.

. . . .

program (or interpreter) generate machine code that can be executed by the machine.[401]

a. Of Front Panels and Blinkenlights

Once computers became programmable and included registers in a central processing unit there still had to be some way to load a program into those registers. Initially this was done by the programmer flipping switches on the front panel of the computer to insert machine code instructions into the computer's registers and reading the contents of those registers by noting the status of the "blinkenlights"[402] on that front panel.

This type of front panel was found both on mainframes like the IBM/System 360 and the MIPS Altair computer, which was the first personal computer.

The switches on the front panel were typically used to "boot" the machine.

---

[401] Although it is usually a good idea for programmers to keep other data separate from the instructions in the program there is nothing in the stored program computer itself that requires that this be done.

[402] Here is an excerpt from the entry on "blinkenlights" from the Jargon File, http://www.catb.org/_esr/jargon/html/B/blinkenlights.html: blinkenlights: /blink'@nli:tz/, n. [common] Front-panel diagnostic lights on a computer, esp. a dinosaur. Now that dinosaurs are rare, this term usually refers to status lights on a modem, network hub, or the like. This term derives from the last word of the famous blackletter-Gothic sign in mangled pseudo-German that once graced about half the computer rooms in the English-speaking world. One version ran in its entirety as follows:

> ACHTUNG! ALLES LOOKENSPEEPERS!
>
> Alles touristen und non-technischen looken peepers! Das computermachine ist nicht fuer gefingerpoken und mittengrabben. Ist easy schnappen der springenwerk, blowenfusen und poppencorken mit spitzensparken. Ist nicht fuer gewerken bei das dumpkopfen. Das rubbernecken sichtseeren keepen das cotten-pickenen hans in das pockets muss; relaxen und watchen das blinkenlichten.

This silliness dates back at least as far as 1955 at IBM and had already gone international by the early 1960s, when it was reported at London University's ATLAS computing site. There are several variants of it in circulation, some of which actually do end with the word 'blinkenlights'. In an amusing example of turnabout-is-fair-play, German hackers have developed their own versions of the blinkenlights poster in fractured English, one of which is reproduced here:

> This room is fullfilled mit special electronische equipment . Fingergrabbing and pressing the cnoeppkes from the computers is allowed for die experts only! So all the 'lefthanders' stay away and do not disturben the brainstorming von here working intelligencies. Otherwise you will be out thrown and kicked anderswhere! Also: please keep still and only watchen astaunished the blinkenlights. The Jargon File is a dictionary of computer jargon that was originally maintained at MIT and that can now be found at http://www.catb.org/jargon/. It is great resource for those of us who are not computer programmers, but who wish to understand how those strange beings think about things.

[The term "boot"] derives from bootstrap loader, a short program that was read in from cards or paper tape, or toggled in from the front panel switches. This program was always very short (great efforts were expended on making it short in order to minimize the labor and chance of error involved in toggling it in), but was just smart enough to read in a slightly more complex program (usually from a card or paper tape reader), to which it handed control; this program in turn was smart enough to read the application or operating system from a magnetic tape drive or disk drive. Thus, in successive steps, the computer 'pulled itself up by its bootstraps' to a useful operating state. Nowadays the bootstrap is usually found in ROM or EPROM, and reads the first stage in from a fixed location on the disk, called the 'boot block'. When this program gains control, it is powerful enough to load the actual OS and hand control over to it.[403]

9. Computers That Can Reprogram Themselves

When we think of computers today, we tend to think not of the dedicated single-purpose computer chips that are hard-wired into devices like toasters and automobiles nor do we think of early machines like the ENIAC[404] that had to be rewired, perhaps using a plug board, each time before they could carry out a new program. Rather we think of general purpose, "programmable" computers,[405] that can, if we just feed them the right instructions in the form of a stream of binary digits—*i.e.*, feed them the right program—perform any computation whatsoever.[406]

It might seem reasonable to restrict our attention only to programmable computers, for we are not concerned with computers without programs. The fact of the matter is, however, that all computers[407] have programs, even if they are embedded in dedicated single-purpose devices. All computers are programmable; the computers that we call "programmable" are more properly referred to as "stored program" computers, for they can store programs as data and can be instructed to reprogram themselves—rewire themselves as it were—in accordance with the instructions contained in the stored

---

[403] *Id.*, http://www.catb.org/~esr/jargon/html/B/boot.html.

[404] *See supra* Part VI.A.4.

[405] *I.e.* "stored program computers" or "Von Neumann machines." *See supra* note 400 and accompanying text.

[406] Provided, of course, that the computation in question is in fact computable. For a discussion of the idea of a computer that can perform any computation whatsoever, *see supra* Part VI.A.3(a)(ii).

[407] That function as computers and not as doorstops or paperweights.

programs.[408] The distinction between the various types of computers would hardly merit mentioning, were it not for the fact that the peculiar claim that software is hardware can be used to justify the Federal Circuit's holding in *Alappat* that a computer program is patentable because a computer program is a machine or a part of a machine[409] can be traced in part to the failure of some legal scholars to recognize that computers that could only be reprogrammed by having someone rewire them still have programs, and that, though those programs may be fixed in the tangible medium of the wiring, the wiring was not the program that runs as a process on the computer any more than a phonograph record is a musical work.

Among those who apparently subscribe to the view that computer programs—at least some computer programs—are parts of a machine are Pamela Samuelson, the leading authority on *soi disant* "intellectual property" rights relating to computer programs, who claimed in her seminal article on copyright and computer programs that "[b]ecause part of the debate on the appropriateness of copyright as a form of protection depends on the answer to the question, 'It is a writing or is it a machine part?,' it is essential to understand programs in their entirety."[410]

Samuelson then goes on to say:

---

[408] The distinction between stored-program computers and other types of computers was carefully made in one of the first articles ever written on computing and the law, John Banzhaf's note about computers and copyright: Digital computers may conveniently be divided into three categories: fixed program, variable program, and stored program. The fixed program machine always performs the same operation upon data of a uniform type. An example would be a cash register which will compute the amount of the sales tax and the customer's change. Since the calculations never change, the mechanism is fixed and it may be said to operate under a fixed program. A variable program (often called a wired program) computer also performs repetitive calculations but these can be altered by changing its wiring. . . . Stored program computers are designed to perform a wide variety of tasks and to permit the pattern of calculations to be changed quickly and easily. . . . Note, *Copyright Protection for Computer Programs*, 64 COLUM. L. REV. 1274, at 1274–75 n. 4 (1964). What Banzhaft calls a "variable program computer" is a computer, like the ENIAC, that could be reprogrammed only by changing its wiring.

[409] *See supra* Part III.A. The most extreme case of confusing a program with wiring is undoubtedly the opinion of the federal District Court in Junger (*c'est moi*) v. Daley, holding that the source code of a computer program is "is a device, like embedded circuitry in a telephone," 8 F. Supp. 2d 708, 713 (N.D. Ohio 1998), *rev'd* 209 F.3d 481 (6th Cir. 2000). The issue in Junger v. Daley was whether "source code" was speech protected by the First Amendment; the 6th Circuit Court of Appeals ultimately rejected the trial judge's holding and held that source code was indeed protected by the First Amendment.

[410] *See supra* Part III.A. The most extreme case of confusing a program with wiring is undoubtedly the opinion of the federal District Court in Junger (*c'est moi*) v. Daley, holding that the source code of a computer program is "is a device, like embedded circuitry in a telephone," 8 F. Supp. 2d 708, 713 (N.D. Ohio 1998), *rev'd* 209 F.3d 481 (6th Cir. 2000). The issue in Junger v. Daley was whether "source code" was speech protected by the First Amendment; the 6th Circuit Court of Appeals ultimately rejected the trial judge's holding and held that source code was indeed protected by the First Amendment.

> It is important to note at the outset that the first generation of computers did not utilize what are now referred to as computer programs to carry out their computational tasks. Rather, they were constructed—or "hard-wired"—in such a way that the machine could perform only the particular function for which it had been wired. To change the operation the machine could accomplish, that is, to enable the machine to perform a different task, the computer's engineers had to rewire or reconfigure the machine. In other words, the first operating computers were all hardware.[411]

And then she inserts the following footnote:

> "To perform different operations, [ENIAC] had to be manually rewired, like an old wire-and-plug telephone switch board, a task that could take several days." Golden, Big Dimwits and Little Geniuses, TIME MAG., Jan. 3, 1983, at 31. One could say that, in a sense, such a machine was programmed by its hardware. That is, the order in which the machine would execute its primitive functions was set by the manner in which the hardware had been constructed. But this is using the word "program" in a very different sense from the way one uses it today to refer to the set of written instructions which are translated to binary code, stored in a computer, and used to cause a computer to perform a useful function.[412]

Note that here Samuelson confuses the meaning of "program" as a noun with its meaning as a verb and then uses the verb "to program" in a most idiosyncratic fashion when she describes the computer as having been programmed by its hardware.[413] Rewiring the ENIAC was the way that it was programmed; the wires by themselves did not do anything, as Banzhaft had made clear[414] twenty years before Samuelson wrote her article.[415]

---

[411] 1984 DUKE L.J. 673–74 (footnote omitted).

[412] *Id.* at 674, n33. It is strange to think that the whole confusion about whether software is hardware may have started with a journalist's off-hand characterization of way the ENIAC was programmed.

[413] For a history of the term "program" as it came to be used in connection with computers, and especially with its use as a verb, see David Alan Grier, *The ENIAC, the Verb "to program" and the Emergence of Digital Computers*, 18 IEEE ANNALS OF THE HISTORY OF COMPUTING 51 (1996). It turns out that the first use of the word "program" as a verb in connection with a digital computer was in paper by John Mauchly, one of the designers of the "ENIAC," the very computer that supposedly was all hardware and therefore did not use programs.

[414] *See supra* note 411.

[415] It should be noted that Samuelson made her claims about programs as hardware as part

Back in the days when the ENIAC was reprogrammed by rewiring it,[416] the programmers gave their written instructions—the program—to the women[417] who were rewiring the computer rather than loading the program into the computer itself. Those early programs, addressed to women, rather than to computers, most certainly were not hardware. Equally clearly, they were written texts and no more patentable than, say, a table of logarithms.

B. Software

> In the old view it is regarded as the purpose of our programs to instruct our machines; in the new one it will be the purpose of our machines to execute our programs.—E. Dijkstra[418]

We have been exploring the nature of computers at some length[419] and in the process we have had to spend a considerable amount of time considering the nature of the computer programs that instruct a computer to do whatever it does. As I have said before, a computer without a program is usable only as a doorstop. The issues that we are concerned with, however, are whether computer programs are

---

of an argument—one that has not been accepted by the courts—that computer programs, at least those in the form of "machine code" should not be protected by copyright. It does not seem likely that she would have used the same argument to reach the conclusion that computer programs should be patentable.

[416] *See* figure accompanying note 402.

[417] Because there was a war going on, both the programmers and those who rewired the ENIAC in accordance with the programmer's programs, were women. A great to-do in those days was made about "Rosie the Riveter"; because, or so I suppose, both military secrecy and the public's incomprehension of what was involved in computing, there was no corresponding recognition of, say, "Polly the Programmer" or "Renata the Rewirer."

[418] Comments at a Symposium, http://tinyurl.com/rq9p8. Edsger Wybe Dijkstra was one of the most influential members of computing science's founding generation. Among the domains in which his scientific contributions are fundamental are:

- algorithm design
- programming languages
- program design
- operating systems
- distributed processing
- formal specification and verification
- design of mathematical arguments

In addition, Dijkstra was intensely interested in teaching, and in the relationships between academic computing science and the software industry. During his forty-plus years as a computing scientist, which included positions in both academia and industry, Dijkstra's contributions brought him many prizes and awards, including computing science's highest honor, the ACM Turing Award. E.W. Dijkstra Archive, *The Manuscripts of Edsger W. Dijkstra*, http://www.cs.utexas.edu/users/EWD/.

[419] Some might say at too much length.

patentable, and whether they should be patentable, without regard to the machines upon which they run. So now the time has come to forget about the machines and concentrate on the programs.

Computer programs are often collectively called "software" and in this article, as you undoubtedly have noticed, I use the terms "computer programs" and "software" pretty much interchangeably.

The term "software" is, however, misleading to the extent that it suggests that software is—that computer programs are—the same sort of "ware" as the hardware—*i.e.*, the computer—upon which it runs. To dispel this confusion it is perhaps enough to note that the term "wetware" is often used by cognitive scientists to refer to the (usually human) brain: it is unlikely that anyone could actually believe that computers and brains and computer programs are all the same sort of "thing", although many a cognitive scientist at least tries to believe that the wetware, the brain, is just a sort of complicated computer on which the programs that we call the "mind" are running.[420]

This distinction between hardware and software involves the distinction between physical objects, on the one hand, and information processing and texts, on the other. If you want to give an old-fashioned cognitive scientist a hard time, just ask him where the programs of the brain are stored when they are not running—and then ask him what the programs are stored as.

That is—or, at least, would be were we doing cognitive science or philosophy—a good question: What are our thoughts stored as when we are not thinking them, when they are not being thought? A text?

The OED 2d defines "ware" as: "A collective term for: Articles of merchandise or manufacture; the things which a merchant, tradesman, or peddler, has to sell; goods, commodities." And in this context the distinction, or lack thereof, between hardware and software takes on considerable legal significance: for example, are computer programs goods that are covered by the Sales Article of the UCC and subject to sales tax? Or are they services? Or are they copyrightable texts or uncopyrightable processes? And—and this is our question—if they are processes, are they patentable processes?

But, whether the usage is felicitous or not, as used here, the term "software" refers to computer programs that are running, or can be run, on a properly programmed computer[421] as well as algorithms

---

[420] *See, e.g.*, J.Z. YOUNG, PROGRAMS OF THE BRAIN (1978).

[421] There is something seemingly circular here that is of critical importance to an understanding of how computers function: programs, with very few exceptions, do not simply run on computers, they are run by—executed by, carried out by—other programs that are already running on the computer. For the insatiably curious though, I should, I suppose, mention that there are little programs—so-called "bootstrap loaders"—that load themselves—like lifting

instructing one—whether one is a programmer or a program—how to create other programs.

By now we should all understand that whatever computer programs may be, they are not devices like the wiring in a telephone or a block and tackle. Computers, at least when they are not human,[422] are indeed devices, but programs are not; programs are either writings or processes, while devices like the innards of a telephone are neither.

Let me repeat that: Computer programs are not devices. Computer programs are either writings or processes. A program is a process when it is being executed by a computer; it is a text when it is just being stored in memory or on a hard disk.

If you have read, or even glanced at, Section 6.1, you should be aware that although computers take many different forms, there is no real dispute as to what computers are: they are devices[423] that we use to assist us in doing computations. On the other hand, there appears for some unknown reason to be lots of confusion about the nature of computer programs.

For our purposes, however, once we distinguish between a computer and the program that runs on the computer[424] it should be clear that it is the computer and the process running on the computer that together assist us in doing computations. The computer by itself

---

themselves by their own bootstraps—into the machine with little or no assistance from other programs except those that are already hard-wired into the computer. The Jargon File includes the following "Historical note" after its discussion of the verb "boot":

> Historical note: this term derives from 'bootstrap loader', a short program that was read in from cards or paper tape, or toggled in from the front panel switches. This program was always very short (great efforts were expended on making it short in order to minimize the labor and chance of error involved in toggling it in), but was just smart enough to read in a slightly more complex program (usually from a card or paper tape reader), to which it handed control; this program in turn was smart enough to read the application or operating system from a magnetic tape drive or disk drive. Thus, in successive steps, the computer 'pulled itself up by its bootstraps' to a useful operating state. Nowadays the bootstrap is usually found in ROM or EPROM, and reads the first stage in from a fixed location on the disk, called the 'boot block'. When this program gains control, it is powerful enough to load the actual OS and hand control over to it. "OS", by the way, stands for "operating system", the program or collection of programs that give the computer its identity as, for example, a Unix box or a MSWindows machine. It is the operating system that runs the various application programs on the computer.

[422] I see no reason, however, to distinguish for present purposes between a computer instantiated as a box containing wires and valves and storage devices, on the one hand, and a computer instantiated as a human being—or a Turing machine, *see supra* Part VI.A.3(a)(ii), for that matter—on the other.

[423] Or people.

[424] We are not concerned here with the computer program as texts, for texts as such are not patentable, although they may, of course, in some cases be copyrighted.

is just a box with some complicated wiring that just sits there if no program is running on it; the program when it is not running on the computer is simply an abstract idea, an algorithm describing the way to carry out a computation.[425]

The computer itself is—if it is not a human being—a machine.[426] The program—*i.e.*, the process—that runs on the machine is, however, a most unusual process, for it can also, at least in theory, be run on a human brain. When the program is run, whether on a machine or in a human brain[427] the result is nothing material, but is rather simply the outcome of a computation: a number or some equally intangible bits of information.[428]

Programs are to computers as thoughts are to brains. It is easy enough these days to think of brains as devices or a collection of devices—but it is not so easy to understand what a thought is, or what the output of a thought process is, or how the brain works. Fortunately we do not have to understand such matters; all we need to know is that, whatever else they do, brains can process information just as computers process information[429] executing exactly the same programs.[430]

Here is a more technical definition of a "computer program" taken from the entry for "Program" in the first edition of the Encyclopedia of Computer Science:

> In order to solve a computation problem, its solution must be specified in terms of a sequence of computational steps, each of which may be effectively performed by a human agent or by a digital computer. Systematic notations for the specification of such sequences of computational steps are referred as programming languages.[431] A specification of the sequence of computational steps in a particular programming language is referred to as a program.[432]

---

[425] That abstract idea or algorithm is most likely stored as a text in some tangible medium like a punch card or a hard disk, but, once again, that does not make the abstract idea patentable.

[426] And is patentable as such.

[427] Assisted, if needed, by pencil and paper.

[428] Although, of course, that information may be stored on a piece of paper or on some other storage device.

[429] Our brains are, after all, capable of performing any operation that can be performed by a Turing machine. *See supra* Part VI.A.3(a)(ii). On the other hand, computations made by brains are slower than those made by a computer and are more likely to contain errors. note that this is not to say that brains cannot do many things that computers cannot do and perhaps will never be able to do.

[430] *See supra* Part VI.A.1.

[431] Programming languages are discussed *infra* Part VI.B.4.

[432] ENCYCLOPEDIA OF COMPUTER SCIENCE 1158 (1976).

A sequence of computational steps, each of which may be effectively performed by a human agent or by a digital computer, is called an "algorithm."[433]

1. Algorithms

Algorithms, and especially mathematical algorithms, are a central concept in mathematics, the philosophy of mathematics, and computer science. On the other hand, algorithms are not often met with in the curricula of our law schools nor in the quotidian practice of "the lawyer in the street"—or even the patent lawyer in the street. Yet we lawyers are never going to understand the actual issues in cases like *Gottschalk v. Benson*[434] unless we have some understanding of what an algorithm is.[435]

Here is the standard definition of algorithms: "Algorithms" are "Precise rules for transforming specified inputs into specified outputs in a finite number of steps".[436] Thus mathematical algorithms are precise rules for solving mathematical problems. But recipes, if they are precise enough, are also algorithms by this definition. And the specification in a patent application, which is supposed to be precise enough that anyone "skilled in the art" can implement it, also is—or, at least, should be—an algorithm.[437]

We have repeatedly made clear that all computer programs are algorithms,[438] but this does not mean that all algorithms are computer programs. In fact, most algorithms are not computer programs.

As Donald Knuth, who wrote—and is still writing—the first and greatest treatise on computer science puts it:

> The notion of an algorithm is basic to all computer programming
>
> . . . .

---

[433] As it was by Justice Douglas in *Gottschalk v. Benson*. *See supra* text accompanying note 38.

[434] In case you have forgotten, *Benson* is the leading case in which the Supreme Court held that algorithms that can be executed by a computer are unpatentable. *See supra* Part II.A.

[435] And what information is.

[436] Donald Knuth, *The Art of Computer Programming*, SEMINUMERICAL ALGORITHMS 181–82 (2d ed., 1981).

[437] The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same . . . . 35 U.S.C. § 112.

[438] *See, e.g.*, *supra* note 21.

> . . . . The modern meaning for algorithm is quite similar to that of recipe, process, method, technique, procedure, routine, except that the word "algorithm" connotes something just a little different. Besides merely being a finite set of rules which gives a sequence of operations for solving a specific type of problem, an algorithm has five important features:
>
> > (1) **Finiteness.** An algorithm must always terminate after a finite number of steps. . . . Note, however, that the number of steps can become arbitrarily large . . . . (A procedure which has all of the characteristics of an algorithm except that it possibly lacks finiteness may be called a "computational method." . . . .)
> >
> > (2) **Definiteness.** Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case. . . . (3) Input. An algorithm has zero or more inputs, *i.e.*, quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects. . . .
> >
> > (4) **Output.** An algorithm has one or more outputs, *i.e.*, quantities which have a specified relation to the inputs. . . .
> >
> > (5) **Effectiveness.** An algorithm is also generally expected to be effective. This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using pencil and paper. . . .[439]

Now it turns out that algorithms can be perceived not only as computer programs but also as the underlying processes that are responsible for Darwinian evolution.

> The theoretical power of Darwin's abstract scheme was due to several features that Darwin quite firmly identified . . . but

---

[439] Donald E. Knuth, *The Art of Computer Programming*, FUNDAMENTAL ALGORITHMS 1–9 (2d ed., 1973) [hereinafter "Fundamental Algorithms"]. *Nota bene* the requirement that the operations of carrying out an algorithm can be performed in principal by a man using paper and pencil corresponds to Justice Douglas's explanation that algorithms can be carried out by "head and hand." *See supra* text accompanying note 76. A man solving a problem by head and hand is a classic example of a "mental process" and his solution to the problem is an idea.

lacked the terminology to describe explicitly. . . . Darwin had discovered the power of an algorithm. An algorithm is a certain sort of formal process that can be counted on—logically—to yield a certain sort of result when it is "run" or instantiated. Algorithms are not new, and were not new in Darwin's day. Many familiar arithmetic processes, such as long division or balancing you checkbook, are algorithms, and so are the decision procedures for playing perfect tic-tac-toe, and for putting a list of words into alphabetical order. What is relatively new—permitting us valuable hindsight on Darwin's discovery—is the theoretical reflection by mathematicians and logicians on the nature and power of algorithms in general, a twentieth century development which led to the birth of the computer, which has led in turn, of course, to a much deeper and more lively understanding of the power of algorithms in general.

The term *algorithm* descends . . . from the name of a Persian mathematician, Mûusâ al-Khowârizm, whose book on arithmetical procedures, written about 835 A.D. was translated into Latin in the twelfth century . . . . The idea that an algorithm is a foolproof and somehow "mechanical" procedure has been present for centuries, but it was the pioneering work of Alan Turing, Kurt Gödel, and Alonzo Church in the 1930s that more or less fixed our current understanding of the term. Three key features of algorithms will be important to us, and each is somewhat difficult to define. . . .

> (1)substrate neutrality: The procedure for long division works equally well with pencil or pen, paper or parchment, neon lights or skywriting, using any symbol system you like. The power of the procedure is due to its logical structure, not the causal properties of the materials used in the insubstantiation, just so long as those causal powers permit the prescribed steps to be followed exactly.
>
> (2) underlying mindlessness: Although the overall design of the procedure may be brilliant, or yield brilliant results, each constituent step, as well as the transition between steps, is utterly simple. How

> simple? Simple enough for a dutiful idiot to perform.
> . . .
>
> (3) guaranteed results: Whatever it is that an algorithm does, it always does it, if it is executed without misstep. An algorithm is a foolproof recipe.
>
> It is easy to see how these features made the computer possible. *Every computer program is an algorithm*, ultimately composed of simple steps that can be executed with stupendous reliability by one simple mechanism or another. Electronic circuits are the usual choice, but the power of computers owes nothing (save speed) to the causal peculiarities of electrons darting about on silicon chips. The very same algorithms can be performed (even faster) by devices shunting photons in glass fibers, or (much, much slower) by teams of people using paper and pencil. . . .[440]

Now it is, I think, fortunate that we do not have to explore Darwinian algorithms or more *outr'e* ideas such as the one that the universe—or, rather, the "multiverse"—is "simply" a quantum computer making every possible computation.[441] The only algorithms that concern us are those that we—or our programs—can write and that can be run on a digital computer or computed by "head and hand."

2. Legal Algorithms

Those reading this who are trained in the law may, upon first hearing the term, conclude that algorithms have nothing to do with legal practice. But every time we draft a contract, or a will, or some proposed legislation, if the instructions that instrument contains are precise enough, then we have just written an algorithm.

Programmers write computer programs; lawyers write forms of legal instruments;[442] both write algorithms. Though lawyers for the most part may not understand—or even care—what computer programs are and though programmers may not know or care what a

---

[440] *Id.*

[441] LLOYD, *supra* note 178.

[442] As lawyers know, but most others perhaps do not, very few lawyers spend their time trying cases in court, but almost all lawyers spend at least some of their time in drafting—and interpreting—legal instruments. As to what a legal instrument is, *see infra* notes 448 and 449 and accompanying text.

legal instrument is, there is little to distinguish one of their activities from the other, even though the texts that they produce are ultimately used to accomplish what at first[443] at least appears to be very different ends.

As you by now should be aware a working definition of a "computer program" is "a set of instructions—that is, an algorithm—that can be fed into a computer in order to cause it to perform certain specified actions." Here are some more definitions that I have liberated from the Jargon File, that collection of often humorous examples of the way that computer programmers think of things;

> 1. A magic spell cast over a computer allowing it to turn one's input into error messages.
>
> 2. An exercise in experimental epistemology.
>
> 3. A form of art, ostensibly intended for the instruction of computers, which is nevertheless almost inevitably a failure if other programmers can't understand it.[444]

It is a bit more difficult to give a definition of a "legal instrument,"[445] but the following definition taken from the Online version of the Oxford English Dictionary should serve well enough for our purposes:

> 5. a. Law. A formal legal document whereby a right is created or confirmed, or a fact recorded; a formal writing of any kind, as an agreement, deed, charter, or record, drawn up and executed in technical form, so as to be of legal validity.

I would add that legal instruments are, as their name suggests, instrumental in accomplishing some goal, in much the same way that

---

[443] As we shall see, legal instruments can be instantiated as computer programs and computer programs can produce forms of legal instruments.

[444] The Jargon File Program.

[445] When I was an undergraduate one of my friends was stabbed, accidentally as it turned out, by a roommate of his who was playing percussion on a radiator with a screwdriver, and who got my friend on the back swing as my friend tried to tap him on the shoulder and tell him to shut up. That had been a busy night for the university police, for a couple of freshmen had earlier gotten into an argument about the temperature at which some home-brewed rocket fuel would explode and had blown up their room when they put their theories to the test. But, in any case, in due time my friend was carted off to the infirmary. The next morning the student newspaper reported that the university police had reported that my friend had been stabbed with a sharp instrument but that the only instrument that they could find in the room was a violin. So perhaps the term "instrument" is inherently ambiguous, although I hope it is clear that a legal instrument is something that is written and that it therefore can't be something like a screwdriver or a violin any more than a computer program is like the wiring in a telephone.

computer programs are; they create legal rights and liabilities and privileges and licenses and instruct the parties to do, or refrain from doing, certain acts.

Perhaps the best example of a legal instrument is a deed that conveys land from one person to another.[446]

Now for a deed to come into existence someone first has to write it—or, in the language of lawyers, draft it—or, perhaps, copy it, with suitable modifications[447] from some pre-existing form. A key point about a draft of a deed is that it is a document, a text, that, by itself does not do anything.[448] It is only when it is executed—that is, signed and dated and delivered by the grantor to the grantee—that it actually becomes an effective legal instrument that conveys the land in question from the grantor to the grantee.

Another example of a legal instrument is the GNU General Public License that applies to this work—or, at least those versions of this work that are computer programs.[449] Perhaps the simplest example of a legal instrument is a promissory note:

> Promissory Note.
>
> FOR VALUE RECEIVED I, Insert Name of Maker , promise to pay to the order of Insert Name of Payee the sum of Insert Amount in Words Dollars ($_____) lawful money of the United States, on demand.
>
> Signature
>
> Date:_____

As has already been mentioned,[450] patent applications are also a form of legal instrument, and one that is supposed to be as precise as any other algorithm.[451]

From this description it should be clear that, when "executed," both legal instruments and computer programs cause something to happen, although the type of things that happen may appear to be

---

[446] A type of legal instrument that I drafted—or just had my secretary copy from an earlier example, filling in the new names of the parties and the description of the land to be conveyed—innumerable times during the nine years that I was a practicing lawyer before I went off to Cleveland and fell among legal academics.

[447] For example, changing the names of the parties and the description of the land

[448] Except perhaps serve as a model—a form—that can be copied and modified to create another draft of a deed.

[449] The GNU General Public License, *available at* http://www.fsf.org/licensing/licenses/gpl.html.

[450] *See supra* note 440 and accompanying text.

[451] Recall that in *Gottschalk v. Benson* the claimed invention was called an "algorithm." *See supra* note 38.

radically different. Computer programs produce numbers—or, more precisely, representations of numbers—as their outputs, while legal instruments produce changes in legal relations. The important point is that neither of these outcomes involves any particular change in any material object. In this respect the processes of writing drafts of legal forms and of writing computer programs are so similar that they should be treated by patent law in the same fashion. If legal forms are not patentable, then computer programs should not be patentable. And if computer programs are patentable, then legal forms should be patentable, a conclusion that should horrify any lawyer who has ever made use of a legal form book.[452]

The near identity of programs and legal instruments was neatly illustrated for me many years ago just after I finished taking the IBM Executive Computer Concepts course.[453] I happened to look into a colleague's office, and there pinned on the wall behind his desk was the largest flow chart I had ever seen.[454] When I asked him what it was for, he reminded me that we represented a settlement agreement involving extensive multiparty litigation and millions of dollars in payments that were to be made by some of the parties to others only if certain conditions were met. It was our job to see that the payments were made when due and to make sure that they were not made when they were not due. He then explained that the flow chart covered all of the possible contingencies that could arise under the agreement, so that he could tell upon the happening of any relevant event exactly who was entitled to what. I was duly impressed and, with my new found interest in computers, asked him if he had thought of implementing the flow chart as a computer program. And then he kindly explained that that would be a complete waste of time since whatever happened would only happen once; that there would never be another agreement like the one whose application we were supervising. And yet, by translating the agreement into that flow chart my colleague had clearly demonstrated the agreement actually was an algorithm and that, when he followed the instructions contained in the flow chart, he was implementing a program that could have been run on a computer were that not a waste of time.

---

[452] It is, in fact, difficult not to conclude that the Federal Circuit has already held in *State Street* that legal forms are patentable, since the "Hub and Spoke Financial Services Configuration" held to be patentable in that case consists of little more than a bunch of legal forms. *See supra* Part III.B.

[453] *See supra* note 385 and accompanying text.

[454] And I have never seen a larger one since.

3. Computational Processes

The simple truth—at least simple to state if not simple to explain—is that the computational processes that are covered by software patents and the *legal processes that are created by legal instruments*. are radically different from processes like curing rubber or building machines or cooking a seagull that are traditional patentable subject matter.

Consider the following passage from the so-called "Wizard Book":[455] the text used in MIT's entry-level computer science course.

> We are about to study the idea of a computational process. Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called data. The evolution of a process is directed by a pattern of rules called a program. People create programs. In effect, we conjure the spirits of the computer with our spells.
>
> A computation process is indeed much like a sorcerer's idea of a spirit. It cannot be seen or touched. It is not composed of matter at all. However, it is very real. It can perform intellectual work. It can answer questions. It can affect the world by disbursing money at a bank or by controlling a robot arm in a factory. The programs we use to control processes are like a sorcerer's spells. They are carefully composed from symbolic expressions in arcane and esoteric programming languages that prescribe the tasks we want our processes to perform.
>
> A computational process, in a correctly working computer, executes programs precisely and accurately. Thus, like the sorcerer's apprentice, novice programmers must learn to understand and to anticipate the consequences of their conjuring. Even small errors (usually called bugs or glitches) in programs can have complex and unanticipated consequences.[456]

---

[455] HAL ABELSON, JERRY SUSSMAN & JULIE SUSSMAN, STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS (MIT Press 1996), is an excellent computer science text used in introductory courses at MIT. So called because of the wizard on the jacket. One of the bibles of the LISP/Scheme world. Also, less commonly, known as the "Purple Book." Jargon File, Wizard Book.

[456] H. Abelson & G. Sussman, *Structure and Interpretation of Computer Programs*, http://mitpress.mit.edu/sicp/full-text/book/book-Z-H-6.html.

Thinking of the computer programs that execute computational processes as the spells of a sorcerer may well be the simplest way for most lawyers to think of them, especially when we realize that we too also cast spells that have very real consequences. When A signs and delivers to B a copy of a deed prepared by a lawyer,[457] suddenly that plot, piece, or parcel of land known as Blackacre belongs to B rather than to A.[458]

As I have pointed out before, few lawyers would think that it is a good idea that the spells in their form books could be patented.[459]

4. Programming Languages

Let me repeat a key part of that quotation from the Wizard Book

> The programs we use to control processes are like a sorcerer's spells. They are carefully composed from symbolic expressions in arcane and esoteric programming languages that prescribe the tasks we want our processes to perform.[460]

We should realize that a patent on a computer program is not limited to any particular programming language,[461] but rather covers all possible implementations written in all possible programming languages. Here, for example, is a program—a very small one—that is written in the BASIC[462] programming language:

```
10 PRINT "Hello World!"

BYE
```

---

[457] Or a lawyer's secretary.

[458] As I recall it, Robert Heinlein, in his short novel Magic, Inc., describes a world like ours in which the laws of magic have replaced the laws of physics; every profession is radically affected by this change except the law, for it turns out that lawyers had been practicing magic all the time.

[459] *See supra* text accompanying note 455.

[460] *See supra* note 459.

[461] Programming languages normally are designed as an abstract set of specifications and then are instantiated as particular programs called "compilers" or "interpreters" running on particular computers. There are lots of versions of the BASIC programming language, many of which run on the same hardware. On the other hand, some compilers, such as the GNU project C and C++ compiler, can compile programs written in more than one programming language.

[462] "BASIC" is an acronym for "Beginner's All-purpose Symbolic Instruction Code." It was developed at Dartmouth for use by non-science students and copying it was encouraged by Dartmouth. In 1975 Microsoft started marketing a version of BASIC as its first product and that version was to become pretty much the standard on personal computers. Wikipedia, *BASIC*. If BASIC had been patented by Dartmouth, we might have been spared "Microsoft Basic," which I at least found to be a totally worthless back in the days before the IBM PC was dreamed of and I was running the CP/M operating system on a NorthStar Horizon computer with 56K of memory. (That is why I had to learn to program using the ASM assembly language. *See supra* note 389).

If one puts that program into a file[463]—in this example, a file named "hellworld"—and then enters the command

% basic hellworld

The characters

Hello World

will appear on the computer's video screen.[464]

A similar "Hello World" program has been written for almost every programming language that exists today,[465] although some of those programs are much more complicated than, and bear little resemblance to, the BASIC version. Thus, for example, the G Code version[466] carves an image on a metal surface, rather than just printing out the characters "Hello World."

Yet each of these multitudinous programs would be covered by a patent on a program that causes "Hello World" to appear on the computer's screen or some other output device. And think of how many programs would be covered by a more general patent covering any program that prints out any string of text on any output device.

---

[463] And on a Unix or a Linux system like mine makes the file executable.

[464] The version of BASIC that I am using here is called "Chipmunk BASIC. *See* BASIC Programming and Chipmunk Basic Home Page, http://www.nicholson.com/rhn/basic/.

[465] For a collection of instructions written in various computer languages that also cause the words "Hello World!" to be printed out on some output device, see "The Hello World Collection," http://www.roesler-ac.de/wolfram/hello.htm.

[466] http://www.roesler-ac.de/wolfram/hello.htm#G-Code. "G Code" is a programming language that is used to control machine tools.

I think that we all should be very happy that, back in the days when the first "Hello World" program was written, nobody dreamed that software could be patented.

Another set of computer programs written in many different programming languages that all produce the same output is the wonderful collection—known as the "Gallery of CSS Descramblers"[467]—of computer programs, assembled and maintained by Dave Touretzky[468]

Here is Touretzky's explanation of why he created the gallery—were it not much too long, I would be delighted to include the entire gallery as an appendix here:

> On January 20, 2000, United States District Judge Lewis A. Kaplan of the Southern District of New York issued a preliminary injunction in *Universal City Studios et al. v. Reimerdes et al.*, prohibiting the defendants from distributing computer code for reading encrypted DVDs. The defendants had been sued under 17 USC 1201(a)(2), also known as section 1201(a)(2) of the Digital Millennium Copyright Act. Judge Kaplan subsequently issued a memorandum order in which he indicated that executable source code was not subject to First Amendment protection against prior restraint of speech. This finding is contrary to that of the 9th Circuit US Court of Appeals, who ruled in the Bernstein cryptography case that source code is indeed protected speech. In their decision, The 9th Circuit even quoted some Scheme code from the declaration of MIT Professor Harold Abelson, explaining why source code is an effective and sometimes preferred means of human communication. Professor Andrew Appel of Princeton University also filed a declaration explaining the importance for computer science of being able to publish source code. More recently, the 6th Circuit US Court of Appeals ruled in the Junger cryptography case[469] that, independent of its functional significance, the expressive nature of source code affords it First Amendment protection.
>
> If code that can be directly compiled and executed may be suppressed under the DMCA, as Judge Kaplan asserts in his

---

[467] http://www-2.cs.cmu.edu/ dst/DeCSS/Gallery/.

[468] Research Professor in the Computer Science Department and the Center for the Neural Basis of Cognition at Carnegie Mellon University, http://www-2.cs.cmu.edu/dst/.

[469] *See supra* note 412.

> preliminary ruling, but a textual description of the same algorithm may not be suppressed, then where exactly should the line be drawn? This web site was created to explore this issue, and point out the absurdity of Judge Kaplan's position that source code can be legally differentiated from other forms of written expression.

Although Touretzky was concerned with the Digital Millennium Copyright Act, his collection nicely illustrates the point that if a software patent[470] is infringed by someone writing, using, or selling a computer program[471] then such infringements would include the writing, executing—either on a machine or by head and hand—, or selling of any algorithm implementing the patented software, even if the algorithm were simply written in English or in the form of a mathematical formula. Or even wearing a tie with the algorithm printed on it.[472]

Touretzky's collection also raises the question of whether it would violate the provisions of the First Amendment protecting the freedoms of speech and of the press to treat the writing or distribution of a computer program as a patent infringement.

5. Software as Text

Having written a few—quite trivial—computer programs, I tend to think of software as a text, in the sense of the first definition of that word given in the online edition of the OED: "The wording of anything written or printed; the structure formed by the words in their order; the very words, phrases, and sentences as written." And thinking of software as a text, I naturally think that it is not patentable and in fact that its publication is protected by the freedoms of speech and of the press that the First Amendment of the constitution assures us shall not be abridged. The fact that software as written is a text—that computer programs as stored in a tangible medium of expression are a text—is so fundamental to an understanding of computing and the law, that there is really not very much that I can say about it.

---

[470] or any other patent.

[471] 35 U.S.C. § 271(a) provides:

> Except as otherwise provided in this title, whoever without authority makes, uses, offers to sell, or sells any patented invention, within the United States or imports into the United States any patented invention during the term of the patent therefor, infringes the patent.

[472] I still have a tie with a "CSS Descrambler" program printed on it. A picture of s tie like that is included in Touretzky's collection.

Programs are written and therefore programs are writings. And writings are texts.

What more is there to say?

There is, of course, the danger that some readers, thinking of programs as processes that run when one "clicks" on an "icon" or a "button," may not be able to accept the reality that software is a written text rather than a machine; the fact remains, however, that every program that can be induced to run by judicious "clicking" is actually stored somewhere within the computer as a text. The use of a Graphical User Interface[473] may make life easier for computer users who have trouble reading and writing, but it also makes it easier to confuse the text of software with a machine. The damage that is done to the minds of those who rely on GUIs to communicate with their computers is explored at length in a essay by Neal Stephenson entitled In the Beginning was the Command Line.[474]

If you have trouble thinking of computer programs as simply being a text, you certainly should read Stephenson's essay. Here is a small sampling of what he has to say:

> [C]omputers do arithmetic on bits of information. Humans construe the bits as meaningful symbols. But this distinction is now being blurred, or at least complicated, by the advent of modern operating systems that use, and frequently abuse, the power of metaphor to make computers accessible to a larger audience. . . . .
>
> People who have only interacted with computers through graphical user interfaces like the MacOS or Windows–which is to say, almost everyone who has ever used a computer–may have been startled, or at least bemused, to hear about the telegraph machine that I used to communicate with a computer in 1973. But there was, and is, a good reason for using this particular kind of technology. Human beings have various ways of communicating to each other, such as music, art, dance, and facial expressions, but some of these are more amenable than others to being expressed as strings of symbols. Written language is the easiest of all, because, of course, it consists of strings of symbols to begin with. If the symbols happen to belong to a phonetic alphabet (as opposed to, say, ideograms), converting them into bits is a trivial procedure, and one that was nailed, technologically, in the

---

[473] *See supra* note 395 and accompanying text.

[474] http://www.cryptonomicon.com/beginning.html.

> early nineteenth century, with the introduction of Morse code and other forms of telegraphy.
>
> We had a human/computer interface a hundred years before we had computers. When computers came into being around the time of the Second World War, humans, quite naturally, communicated with them by simply grafting them on to the already-existing technologies for translating letters into bits and vice versa: teletypes and punch card machines.
>
> . . . .
>
> [T]he first job that any coder needs to do when writing a new piece of software is to figure out how to take the information that is being worked with (in a graphics program, an image; in a spreadsheet, a grid of numbers) and turn it into a linear string of bytes. These strings of bytes are commonly called files or (somewhat more hiply) streams. They are to telegrams what modern humans are to Cro-Magnon man, which is to say the same thing under a different name. All that you see on your computer screen–your Tomb Raider, your digitized voice mail messages, faxes, and word processing documents written in thirty-seven different typefaces–is still, from the computer's point of view, just like telegrams, except much longer, and demanding of more arithmetic.

When discussing software by itself we may find it helpful to consider it to be patterns or data or information. But when we discuss the legal issues relating to software the simplest way to think of it, at least initially, is as a text—as something that is written that can be copied and communicated—or as a process that produces such a text.

It was, perhaps not surprisingly, one of the most insistent advocates of the peculiar idea that a text can be a machine who was forced to confront the question of whether a patent on a machine made of text could infringe the freedom of speech and of the press clauses of the First Amendment, for it is the communication and printing of texts that is protected by that amendment and, even if one believes that a text is a patentable machine, a text still remains a text.

Back in November of 2000 Dan L. Burk published an article entitled Patenting Speech[475] in which he claimed that:

---

[475] 79 TEX. L. REV. 99 (2000).

> Far from being the instruction manual to an intricate and complicated machine, computer code is in fact the machine itself. Software is not a text, it is a machine built of text. Just as physical machines are built from tangible media such as wood, steel, and plastic, programs are built from source code.[476]

Of course, having concluded that the text that is a computer program is also a patentable machine, Burk was ultimately forced to deal with the problem of whether the idea—the algorithm—contained in the text of a program was patentable if it were implemented by a human being rather than by a computer and, if the idea were patentable, how that result could be justified under the terms of the First Amendment.

Here is a small part of what Burk has to say on this point:

> To say that software has been recently accommodated within patent law is not to say that software now fits comfortably within patent law. If the functional characteristics of software that fit poorly within copyright seem to indicate it would be better protected under patent law, the expressive characteristics that seem to indicate software is better protected under copyright concomitantly fit poorly within patent law. Initially, the Supreme Court seemed to indicate in *Gottshalk v. Benson* that software algorithms could not be protected under patent law. The Court reached much the same result in *Parker v. Flook*, but subsequently modified its rule in *Diamond v. Diehr* to hold that software could be patentable under rather stringent constraints. In particular, the Supreme Court specified that the operation of the computer program must be tied to some physical result—almost a "fixation" rule for patent law.
>
> This requirement of a physical result linked to the computer process was at first applied with some rigor. However, lower courts, particularly the Court of Appeals for the Federal

[476] *Id.* at 119. In fairness to Burk it should be pointed out that he was criticizing cases, like *Junger v. Daley* (*see supra* note 412), where the "source code" written by a programmer was treated as a text protected by the First Amendment, while the "machine code" that is executed by a computer was not accorded that protection. (Although whether machine code is protected by the First Amendment was not at issue in those cases). As Touretzky's Gallery of CSS Descramblers demonstrates (*see supra* note 470 and accompanying text), it makes as little sense to treat source code as a text while denying the same status to machine code as it does to claim that such texts are actually machines.

> Circuit, have incrementally stretched the *Diehr* holding to the point that patent now covers essentially any sort of software, in almost any embodiment. Thus, . . . [it] may be that we now have a category of patentable expression in computer software, a juxtaposition of legal requirements that is wholly unprecedented. Moreover, the proliferation of software-related patents for systems that traditionally constitute forms of speech, including systems of writing and systems for voice command recognition, has created a unique set of problems. Unlike copyright, which was intended to deal with expressive subject matter, patent law has been almost entirely isolated from First Amendment considerations. If copyright law has proven poorly suited to distinguishing function from expression, patent law is likely to prove even more poorly suited to the task, and may lack altogether the doctrinal tools to accommodate the First Amendment concerns associated with proprietary speech.[477]

Of course, as Burk seems to realize,[478] the difficulties with applying First Amendment protections to patented speech would simply disappear were the courts to return, as I think they inevitably will, to the rule established in *Benson* that algorithms are not patentable. Even if one suffers from the delusion that a computer program is a machine made out of text that does not necessarily mean that computer programs should be patentable. As Burke says:

> If patent law is unable to separate the expressive and functional aspects of computer code, then it may instead be required to somehow accommodate First Amendment interests in that code, much as copyright law has done in its development of fair use, original authorship, and the idea/expression dichotomy. But patent law at present is singularly ill-equipped to make such an accommodation, particularly now that it has been reshaped to protect computer algorithms. It is no coincidence that the admission of software into the canon of patentable subject matter has resulted in the systematic eradication of a cluster of patent doctrines, for it was precisely these doctrines restricting the patentability of mental steps or printed matter that tended to impede the incorporation of software into patentable subject

---

[477] *Id.* at 137 (footnotes omitted).

[478] In the next Part of his article Burk gives has an excellent discussion of the demise of the doctrine that "mental steps" are not patentable at the hands of the Federal Circuit.

> matter. The introduction of expressive subject matter into patent law may require revival or reformation of one or more of these patent doctrines in order to re-establish the line between function and expression in patentable subject matter. Alternatively, some doctrine of patent fair use may be required to accommodate expressive material that falls under the patent umbrella.[479]

6. Software as a Thought Process

As I have mentioned,[480] those who have not written a computer program or two are likely to think of computer programs as processes that run when one clicks on an the icon that invokes them. Thus we click on the icon for a calculator and asks it what is 328÷3 and, after an extremely short while, it prints out 328÷3 =109.333333333, doing our thinking for us. Or we submit the word "wubmit" to a spelling checker which informs us that we probably meant "submit," again doing our thinking for us.[481]

If when we solve those problems in our heads—perhaps with the aid of paper and pencil or a dictionary—we think that we are thinking, why should we deny that thought process are involved when we ask a computer—or our secretary—to solve the problems for us?

In the preface to the first edition of the text used in the entry level course in computer science at MIT, the authors say:

> Our design of this introductory computer-science subject reflects two major concerns. First, we want to establish the idea that a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute. . . .
>
> . . . .
>
> Underlying our approach to this subject is our conviction that "computer science" is not a science and that its significance has little to do with computers. The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is the emergence

---

[479] *Id.* at 160–61.

[480] *See supra* note 477 and accompanying text.

[481] I was surprise to note that my spell checker produced "wabbit" as another possibility.

> of what might be called procedural epistemology—the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Mathematics provides a framework for dealing precisely with notions of "what is." Computation provides a framework for dealing precisely with notions of "how to."[482]

Viewed in this fashion, of course, programs have little to do with computers. Nor are they are functional in the way that a machine is functional. If they are tools then they are tools for thought. And in that case it is hard to imagine how anyone could argue that tools for thought should be patentable.[483]

Now such tools for thought are, at the moment, pretty much limited to two distinct categories: programs that run on a brain and programs that run on a computer. But the day is not far away, if it has not already arrived, when a computer on a chip will be inserted in a human brain.[484]

If someone were to argue, as someone probably will, that if there is a patent on a computer program it would be infringed if one, who is not licensed to use it, executes it on a computer, but not infringed if one executes it in one's brain, what happens when a computer is wired into a brain?

It is hard not to agree with the Supreme Court's opinion in *Gottschalk v. Benson* that algorithms that describe mental processes are not patentable, whether they are executed by a computer or by "head and hand."

7. Interchangeability of Software and Data

It is customary to distinguish between programs and data, programs containing the instructions as to how the data is to be processed, data being the information that is processed by the program running on the computer. In so far as I know, no one has ever argued that the data processed by a computer is a machine, or part of a machine, or so functional that it should not be treated as text. In actuality, however, there is no absolute way to distinguish a

---

[482] H. ABELSON & G.J. SUSSMAN, STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS xvii–xviii (2d ed., 1996) [hereinafter ABELSON].

[483] See also what Newell had to say in the text *supra* accompanying note 196.

[484] *See* Bob Calverley, *USC Engineers Look to the Brain For the Next-Generation Computer Chip*, http://www.usc.edu/uscnews/stories/8213.html. Making a chip that can be connected to human brain tissue and take over a cognitive function that has been destroyed by ailments such as epilepsy and Alzheimer's is one researcher's ultimate goal.

program from data. For example, the source code of a program written in the C programming language is the data that is processed by a C compiler that converts the source code into object code.[485]

8. Software as Fundamental Truths or Laws of Nature

Since the Church-Turing thesis tells us that a Turing Machine, or any general purpose computer, can be used to calculate any calculable

---

[485] And the object code that is produced by that compiler is the data—the source code—that is processed by a decompiler that attempts to turn what was the object code back into something like the original source code.

Here is a more technical discussion of this point, taken from the Part of Abelson that is captioned: "Data as Programs:"

In thinking about a Lisp program that evaluates Lisp expressions, an analogy might be helpful. One operational view of the meaning of a program is that a program is a description of an abstract (perhaps infinitely large) machine. For example, consider the familiar program to compare factorials:

(define(factorial n) (if (= n 1) 1 (* (factorial (- n 1)) n)))

We may regard this program as the description of a machine containing parts that decrement, multiply, and test for equality, together with a two position switch and another factorial machine. (The factorial machine is infinite because it contains another factorial machine within it.) . . . . In a similar way, we can regard the evaluator as a very special machine that takes as input a description of a machine. Given this input, the evaluator configures itself to emulate the machine described. For example, if we feed our evaluator the definition of a factorial . . ., the evaluator will be able to compute factorials. From this perspective, our evaluator is seen to be a universal machine. It mimics other machines when these are described as Lisp programs. This is striking. . . . Another striking aspect of the evaluator is that it acts as a bridge between the data objects that are manipulated by our programming language and the programming language itself. Imagine that the evaluator program (implemented in Lisp) is running, and that a user is typing expressions to the evaluator and observing the results. From the perspective of the user, an input expression such as (* x x) is an expression in the programming language, which the evaluator should execute. From the perspective of the evaluator, however, the expression is simply a list (in this case, a list of three symbols: *, x, and x) that is to be manipulated according to a well-defined set of rules. That the user's programs are the evaluator's data need not be a source of confusion. In fact, it is sometimes convenient to ignore this distinction and give the user the ability to explicitly evaluate a data-object as a Lisp expression . . . . ABELSON, *supra* note 382, at 384–87 (footnotes omitted). The authors add the following information in a footnote: The fact that the machines are described in Lisp is inessential. If we give our evaluator a Lisp program that behaves as an evaluator for some other language, say C, the Lisp evaluator will emulate the C evaluator, which in turn can emulate any machine described as a C program. Similarly, writing a Lisp evaluator in C produces a C program that can execute any Lisp program. The deep idea here is that any evaluator can emulate any other. Thus, the notion of "what can in principle be computed" (ignoring practicalities of time and memory required) is independent of the language or the computer, and instead reflects an underlying notion of computability. This was first demonstrated in a clear way by Alan Turing (1912–1954), whose 1936 paper laid the foundations for theoretical computer science. In the paper, Turing presented a simple computational model—now known as a Turing machine—and argued that any "effective process" can be formulated as a program for such a machine. (This argument is known as the Church-Turing thesis.) Turing then implemented a universal machine, *i.e.*, a Turing machine that behaves as an evaluator for Turing machine programs. He used this framework to demonstrate that there are well-posed problems that cannot be computed by Turing machines . . ., and so cannot be formulated as "effective processes." Abelson, *supra* note 382, at 386, n 19. For more on Turing machines *see supra* Part VI.A.3(a)(ii).

function, it follows that when any such calculation is made it will always produce the same result and that that outcome is necessarily a fundamental[486] truth or law of mathematics. Thus when one performs a computation, with or without the aid of a computer, the result that one "discovers" will inevitably be a fundamental[487] truth or law, not only of mathematics, but also of nature.[488]

This, of course, is the basis for the conclusion of the Supreme Court in *Gottschalk v. Benson* that mathematical algorithms cannot be patented because they are "laws of nature."[489]

9. Software Revisited

There is a wonderful article by Peter Suber[490] entitled "What is Software?"[491] that anyone who wants to grasp the true nature of software should read.

The key points that Suber makes are summed up in the "Abstract" at the beginning of the article:

> In defining the concept of software, I try at first to distinguish software from data, noise, and abstract patterns of information with no material embodiment. But serious objections prevent any of these distinctions from remaining stable. The strong thesis that software is pattern per se, or syntactical form, is initially refined to overcome obvious difficulties; but further arguments show that the refinements are trivial and that the strong thesis is defensible.

What is of most interest to me is the fact that Suber could not draw a strong distinction between "software" and "abstract patterns of information," that is, that he could not make a strong distinction between software and the abstract information that could be processed by a computer in accordance with the instructions contained in the software.

---

[486] If perhaps trivial.

[487] *See supra* note 489.

[488] The fact that most of what we call "laws of nature" can only be established inductively, while the truths of mathematics are arrived at deductively is not a distinction that need concern us here, any more than we need to take a position on the debates among philosophers of mathematics about whether the realists or the formalists or the inductivists are right. (If you are interested in such issues, you might want to look at Wikipedia, *Philosophy of Mathematics, Contemporary Schools of Thought*, http://tinyurl.com/pp8u7).

[489] *See supra* text accompanying note 40.

[490] One of the few people who have taught both philosophy and computer science and to have both a JD and a PhD. http://www.earlham.edu/_peters/hometoc.htm.

[491] http://www.earlham.edu/_peters/writing/software.htm.

It turns out that software is information[492] on how to process data and data is information[493] that someone is interested in processing.

And that leads, inevitably, the question: What is information?

C. Information

The question: "What is information?" turns out to be quite difficult to answer. People use the word "information" with a variety of meanings and, more often than not, they assume that it must mean something or be informative in some way. The use of the word "information" that concerns us, however, is its use in the description of computers as machines[494] that do nothing but implement algorithms[495] that mindlessly process information without any concern for meaning or informativeness.[496]

For our purposes "information" is a pattern of bits[497] that can be stored on a tangible substrate like a CD-ROM or transmitted as a signal conducted by a stream of electrons or photons or other forms of energy. At a more fundamental level, where fortunately we need not go, matter, energy, and information are the fundamental categories that make up the universe that is studied by physicists, although information has not been as extensively studied as matter and energy.[498]

Suber was not quite right when he said that software, which as we have seen is a form of information, is "patterns of information." Information does take the form of patterns all right, but it would obviously lead to an infinite regress to say that information consists of patterns of information which are patterns of information which are patterns of *informationund so weiter*. Instead information consists of a pattern or patterns exhibited by particles of matter or by quanta of energy.

---

[492] Or "patterns of information."

[493] Including software.

[494] Or people.

[495] *See supra* Part VI.B.1.

[496] Information tends to drive out knowledge. Information is just signs and numbers, while knowledge has semantic value. What we want is knowledge, but what we often get is information. It is a sign of the times that many people cannot tell the difference between information and knowledge, not to mention wisdom, which even knowledge tends to drive out. HEINZ PAGELS, THE DREAMS OF REASON 49 (Simon and Schuster 1988).

[497] *I.e.* binary digits.

[498] General systems theorists often refer to matter, energy and information as fundamental categories. The three concepts—matter, energy and information—are related through scientific laws. Matter and energy relations are more thoroughly understood than relations involving information. At the level of data or signal "difference" is suggested as a more elementary term than "information." Stuart A. Umpleby, *Physical Relationships Among Matter, Energy and Information*, http://tinyurl.com/q2vfl.

Information in the sense that we are using it—as well as the use of the term "bit" for binary digit—was first well defined in C.E. Shannon's seminal article "A Mathematical Theory of Communication."[499]

As Shannon explains at the outset:

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have meaning; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. . . .[500]

Shannon was concerned with the amount of information—the number of meaningless bits—that it would to take to convey a message from one point to another and it is meaningless bits in this sense that are processed by a computer, often to satisfy someone's desire to compress the number of bits in a message.

There is thus some sort of relation between the concepts of bits and of information, although what that relation *is* is not easy to explain.[501] But then I am not alone in having this difficulty. Seth Lloyd writes in his book "Programming the Universe":

> I began the initial meeting of my MIT graduate course on information in the manner I begin all of my courses: "First," I said to the twenty-odd students, "you ask questions and I'll try to answer them. Second, if you don't ask questions, I'll ask you questions. Third, if you don't answer my questions, I'll tell you something I think you ought to know. Any questions?"
>
> I waited. No response.
>
> Something was wrong. Normally, MIT students are more than happy to try to stump the professor, particularly if the alternative is that the professor will try to stump them.

---

[499] 27 BELL SYSTEM TECHNICAL JOURNAL 379–423, 623–56 (1948). You may recall that it was also Claude Shannon who first recognized that Boolean algebra can be used to describe the circuits within digital electronic computers. *See supra* notes 358–61 and accompanying text.

[500] *Id.* at p. 1.

[501] I am reminded of the unhappy days when I was a draftee in the infantry and our first sergeant used to say every other day or so at some formation or other: "Gentlemen, I want to tell you something I know nothing about."

> I moved on two step two: "No questions? Then here's one for you: What is information?"
>
> Nothing. This was even worse. After all, these students had been stuffing themselves full of information since freshman year. If they didn't regurgitate some of it, I was going to have to resort to step three.
>
> "OK. How about this one: What is the unit of information?"
>
> At once, the class responded, "The bit!"
>
> What do my students' answers, or lack thereof, reveal? That it is far easier to measure a quantity of information than to say what information is. . . ."[502]

It is Lloyd's thesis that the universe is a computer—a quantum computer—that computes itself, and this leads him to discuss many difficult matters that we do not need to consider. But Lloyd's initial discussions of what information is and what bits are and what computers *are* are very relevant to the matters discussed in this article and it certainly should not hurt you to read them.

The key point for our purposes is that information is not composed of the physical particles or quanta that may embody or carry it; rather information is the order—or pattern—that may be exhibited by those—or other—particles and quanta. Information is not matter or energy, it is simply a pattern, even though there could be no pattern were there no matter or energy to be patterned. Information is what a text contains, it is not the ink and paper—or other matter—which compose the substrate in which the information is stored. The word "cat" is not a cat, nor are the letters "c", "a", and "t" a cat. The word "cat" is information that we English speakers are likely to interpret as a name that refers to a member of the species *felix cattus*. A picture of a cat is not a cat, although we may interpret it as conveying information about a cat, or perhaps about cats in general. I have a pretty good idea of what a cat is, but that idea is not a cat—it, too, is information—a pattern of some sort in my brain—and not any material thing. As a poet once put it,

> Between the idea
> And the reality
> Between the motion

---

[502] LLOYD, *supra* note 178, at 17.

And the act
Falls the Shadow

The grin on the face of a cat is not the cat. it is pure information. It is the cat's expression. On the other hand, that expression cannot exist without the cat, just as no other information can exist without a material or energetic substrate. The idea of somehow separating an expression from its substrate inevitably reminds me of the Cheshire Cat in "Alice in Wonderland" who vanished leaving only his grin behind.

> . . . and this time it vanished quite slowly, beginning with the end of the tail, and ending with the grin, which remained some time after the rest of it had gone.
>
> The Cheshire Cat

But I suppose that I should stop this before I get busted for practicing philosophy without a license.

The key points are that information is not any material thing, nor is it an article of manufacture and that, on the other hand, information cannot exist apart from a material[503] substrate.

VII. CONCLUSION

When I was in law school back in the mid-fifties of the last century, the received—if somewhat oversimplified—wisdom, even among those of us who wanted nothing to do with patent law, was that all patent claims that have been reviewed by the Supreme Court

[503] or energetic.

are invalid and that all patent claims that have not been reviewed by the Supreme Court are valid.

I gather that nothing has really changed since then, except that the Court had, until recently, pretty much given up reviewing patent claims, And the result, of course, has been that all sorts of patent claims—including those that cover software—have been upheld by the lower courts—in particular by the Federal Circuit that has of late had nearly exclusive jurisdiction over appeals in patent cases, even though the Supreme Court had earlier held in the cases known as *Benson*, *Flook* and *Diehr* that software, including algorithms for the processing of information and mathematical formulae, is, like other abstract ideas, not patentable.

The Supreme Court, however, has of late evidenced a willingness once again to review patent cases and has even raised sua sponte the question of whether a patent was invalid because one cannot patent "laws of nature, natural phenomena, and abstract ideas." And it is also no longer true that the Federal Circuit can hear appeals in patent cases where the issue of the validity of the patent was not initially raised in the plaintiff's complaint: such appeals in federal cases are now heard by the numbered or D.C. Circuits.

It seems improbable that we will have to wait very long before the Court has, once again, the opportunity to consider the validity of a patent on a computer program.

It also seems improbable that when that day of judgment comes the Court will refuse to follow its own holdings in *Benson*, *Flook*, and *Diehr* and will instead treat those cases as having been overruled by the judges of the Federal Circuit. The Court has, after all, never taken kindly in the past to efforts by the lower courts to openly refuse to follow its precedents.

On the other hand, it seems equally certain that those members of the patent bar and legal academics, who have for years been advising their clients and students in reliance on the opinions of the Federal Circuit that software is patentable, will raise a mighty protest that the Supreme Court was wrong when it decided *Benson*, *Flook*, and *Diehr* and that the Federal Circuit was right in acting as if it had overruled those cases.

The trouble with that position is that, as I argue in this article at what most of you will consider excessive length, the Supreme Court was right in holding that computer programs are no more patentable than are mathematical inventions like the calculus or logical truths like De Morgan's law that "NOT (A AND B)" equals "NOT A OR NOT B". Computer programs are texts, not machines as some

lawyers have confused themselves into believing, and thus they may be copyrighted and protected by the First Amendment, but they are not patentable as machines. Computer programs are indeed processes, but they are not patentable processes because what they process is information and what they produce is information, not some modification of material goods or articles of commerce.

The simple fact is—though the reasons for it may be hard for most lawyers to grasp—that, as the title of this article puts it: "You can't patent software: patenting software is wrong."